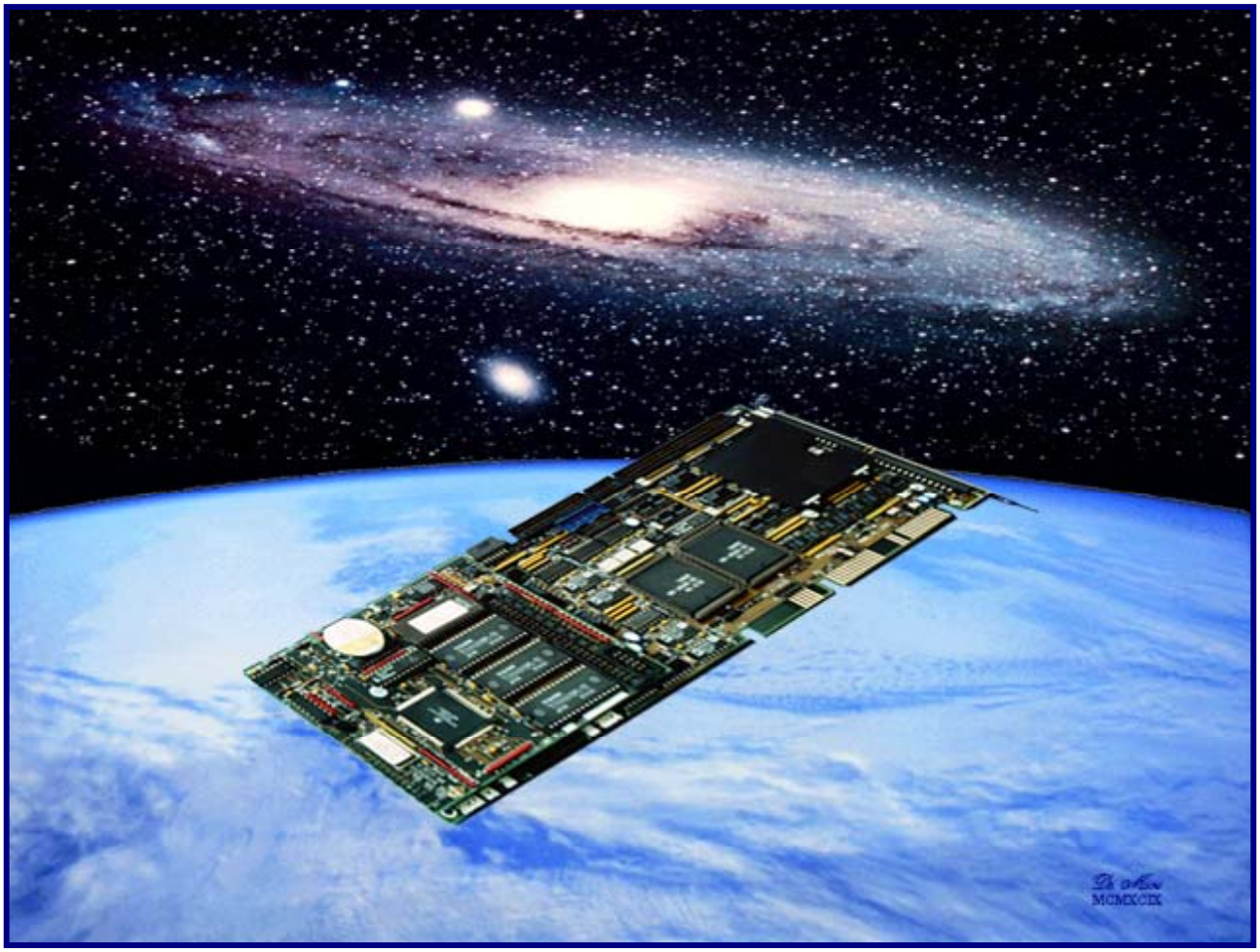
DELTA TAU
Data Systems, Inc.

PMAC

Programmable Multi-Axis Controller



Quick Reference

Fall 2002

Power // Flexibility // Cost Effectiveness // Customer Service // Ease of Use

21314 Lassen Street Chatsworth, CA 91311 // Tel. (818) 998-2095 Fax. (818) 998-7807 // www.deltatau.com

Copyright Information

© 2003 Delta Tau Data Systems, Inc. All rights reserved.

This document is furnished for the customers of Delta Tau Data Systems, Inc. Other uses are unauthorized without written permission of Delta Tau Data Systems, Inc. Information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support
Phone: (818) 717-5656

Fax: (818) 998-7807

Email: support@deltatau.com

Website: http://www.deltatau.com

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

1.0 - INTRODUCTION TO PMAC ____ 1

1.1 - About this manual ____ 1

1.2 - Description of PMAC ____ 1

1. 3 - Types of PMAC ____ 2
1.3.1 - PMAC(1) PC or PMAC(1) VME features ____ 2
1.3.2 - PMAC PC ____ 2
1.3.3 - PMAC *Lite* ____ 2
1.3.4 - PMAC VME ____ 2
1.3.5 - PMAC STD ____ 2
1.3.6 – *Mini*-PMAC ____ 4
1.3.7 – PMAC2 ____ 4
1.3.8 – PMAC2 Ultralite ____ 4
1.3.9 – Turbo PMAC family ____ 4

1.4 – PMAC(1) connectors and indicators ____ 5
1.4.1 – Display Port Outputs (JDISP Port) ____ 5
1.4.2 – Control-Panel Port I/O (JPAN Port) ____ 5
1.4.3 – Thumbwheel Multiplexer Port I/O (JTHW Port) ____ 5
1.4.4 – Serial Port Connection ____ 5
1.4.5 – General-Purpose Digital Inputs and Outputs (JOPTO Port) ____ 5
1.4.6 – Machine Connectors ____ 5
1.4.7 – LED Indicators ____ 5

1.5 - Working with PMAC ____ 6
1.5.1 - Hardware Setup ____ 6
1.5.2 - Software Setup ____ 6
1.5.3 - Programming PMAC ____ 6

1.6 - PMAC Tasks ____ 7
1.6.1 - Single Character I/O ____ 8
1.6.2 - Commutation Update ____ 8
1.6.3 - Servo Update ____ 8
1.6.4 -VME Mailbox Processing ____ 8
1.6.5 - Real-Time Interrupt Tasks ____ 9
1.6.6 - Background Tasks ____ 9
1.6.7 - Observations: ____ 10
1.6.8 - Priority Level Optimization ____ 11

2.0 - PMAC EXECUTIVE PROGRAM, PEWIN ____ 13

2.1 - Configuring PEWIN ____ 13

2.2 - Quick Plot feature ____ 14

2.3 - Saving and retrieving PMAC parameters ____ 14

2.4 - The WATCH and POSITION windows 15

2.5 - Uploading and Downloading files 15

2.6 - Using MACRO names and Include Files 15

2.7 - Downloading compiled PLCCs 15

2.8 - The PID Tuning Utility 16

2.9 - Other Features 18

3.0 - INSTALLING AND CONFIGURING PMAC 19

3.1 - Jumpers Setup 19

3.2 - Serial Connections 19

3.3 - Establishing Host Communications 20
3.3.1 - Terminal Mode Communications 20
3.3.2 - Resetting PMAC for first time use 20

3.4 - Connections 21

3.5 - Power Supplies 21
3.5.1 - Digital Power Supply 21
3.5.2 - Analog Power Supply 21
3.5.3 - Flags Power Supply (optional) 21

3.6 - Overtravel limits and Home switches 21
3.6.1 - Disabling the overtravel limits flags 22
3.6.2 - Types of overtravel limits 22
3.6.3 - Home switches 22
3.6.4 - PMACPack and PMAC2 flag inputs 23
3.6.5 - Checking the flag inputs 23

3.7 - Motor signals connections 23
3.7.1 - Incremental Encoder Connection 23
3.7.2 - Checking the encoder inputs 24
3.7.3 - Checking the DAC outputs 24
3.7.4 - DAC Output signals 24
3.7.5 - Amplifier enable signal (AENAx/DIRn) 25
3.7.6 - Amplifier fault signal (FAULTn) 25
3.7.7 - General-Purpose Digital Inputs and Outputs (JOPTO Port) 25

3.8 - Machine Connections Example 26

3.9 - Software Setup 27

4.0 - PROGRAMMING PMAC 29

4.1 - Online Commands 29

4.2 - Buffered (Program) Commands 30

4.3 - COMPUTATIONAL FEATURES 30
4.3.1 - I-variables 30

4.3.2 - P-Variables 30
4.3.3 - Q-Variables 31
4.3.4 - M-Variables 31
4.3.5 - Array capabilities 32
4.3.6 - Operators 33
4.3.7 - Functions 33
4.3.8 - Comparators 34
4.3.9 - User-Written phase and User-Written servo algorithms 34

4.4 - Memory Map 34
4.4.1 - User Buffer Storage Space 35

4.5 - Encoder Conversion Table 35
4.5.1 - Conversion Table Structure 36
4.5.2 - Further Position Processing 36

4.6 - PMAC position registers 36

4.7 - Homing Search Moves 38

4.8 - Command and Send statements 39

5.0 - MOTION PROGRAMS 41

5.1 - How PMAC Executes a Motion Program 41

5.2 - Coordinate Systems 42
5.2.1 - Axis definitions 42
5.2.2 - Axis Definition Statements 42

5.3 - Writing a MOTION PROGRAM 43

5.4 - Running a MOTION PROGRAM 44

5.5 - Subroutines and Subprograms 45
5.5.1 - Passing Arguments to Subroutines 45
5.5.2 - G, M, T, and D-Codes (Machine-Tool Style Programs) 45

5.6 - Linear blended moves 46
5.6.1 - Observations: 47

5.7 - Circular Interpolation 50

5.8 - Splined Moves 52

5.9 - PVT-Mode Moves 52

5.10 - Other programming features 54
5.10.1 - Rotary Motion Program Buffers: 54
5.10.2 - Internal Timebase, the feedrate override 54
5.10.3 - External Time-Base Control (Electronic Cams) 54
5.10.4 - Position Following (Electronic Gearing) 55
5.10.5 - Cutter Radius Compensation 55
5.10.6 - Synchronous M-Variable Assignment 55
5.10.7 - Synchronizing PMAC to Other PMACs 55
5.10.8 - Axis Transformation Matrices 55
5.10.9 - Position-Capture and Position-Compare Functions 55
5.10.10 - Learning a Motion Program 55

6.0 - PLC PROGRAMS 57

6.1 - Entering a PLC Program 58

6.2 - PLC Program Structure 58

6.3 - Calculation Statements 58

6.4 - Conditional Statements 59
6.4.1 - Level-Triggered Conditions: 59
6.4.2 - Edge-Triggered Conditions: 59

6.5 - WHILE Loops 59

6.6 - COMMAND and SEND statements 60

6.7 - Timers 60

6.8 - Compiled PLC Programs 61

7.0 - TROUBLESHOOTING SECTION 63

7.1 - Resetting PMAC to factory defaults 63

7.2 - The watchdog timer (red LED) 63

7.3 - Establishing communications 64
7.3.1 - General 64
7.3.2 - Bus Communications 64
7.3.3 - Serial communications 64

7.4 - Motor parameters 65

7.5 - Motion programs 65

7.6 - PLC programs 66

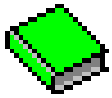
Appendix 1: PMAC error code summary
Appendix 2: PMAC I-variables summary
Appendix 3: PMAC On-Line (Immediate) Commands
Appendix 4: PMAC program command specification
Appendix 5: Motor Suggested M-variable Definitions
Appendix 6: I/O Suggested M-variable Definitions
Appendix 7: ACC-8D / 8P pin-out description
Appendix 8: PMAC Accessories and Options

1.0 - Introduction to PMAC

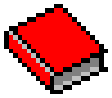
1.1 - About this manual

This manual is intended for first time users of the PMAC motion control. It is oriented to the PMAC(1) family of motion controls and it does not cover other PMAC families: PMAC2, Turbo PMAC(1), Turbo PMAC2, MACRO or UMAC. The subjects illustrated could be used as a quick informative features description or as a roadmap for a more advanced learning through the main documentation. It is strongly recommended to use this quick reference manual in conjunction with the following manuals:

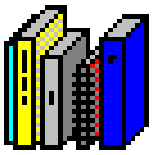
Software Reference



User's Manual



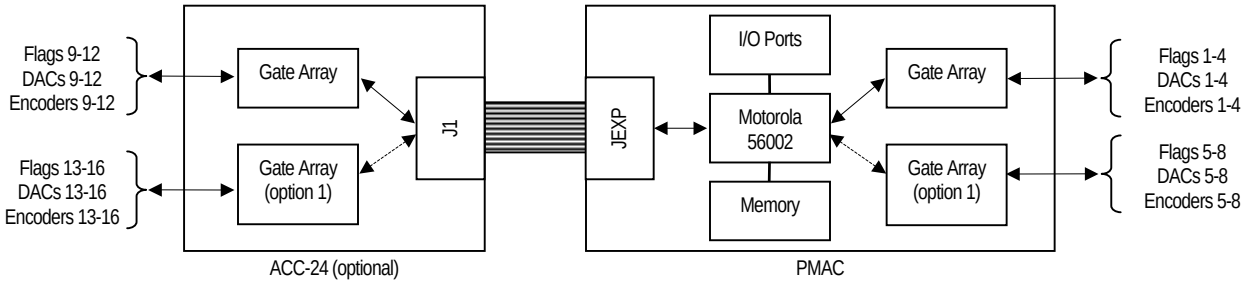
Hardware References



Accessory Manuals

1.2 - Description of PMAC

PMAC, pronounced *"Pe'-MAC"*, stands for Programmable Multi-Axis Controller. It is a family of high-performance servo motion controllers capable of commanding up to eight axes of motion simultaneously with a high level of sophistication.



- There are five hardware versions of PMAC: the PMAC-PC, the PMAC-*Lite*, the PMAC-VME, the PMAC-STD and the *Mini*-PMAC. These cards differ from each other in their form factor, the nature of the bus interface, and in the availability of certain I/O ports.
- Motorola's Digital Signal Processor (DSP) DSP56k is the CPU for PMAC, and it handles all the calculations for all eight axes.
- The registers in PMAC's "DSPGATE" Gate-Array ICs are mapped into the memory space of PMAC's processor. Each DSPGATE contains four consecutively numbered channels; there may be up to 4 DSPGATEs in a PMAC system, for up to 16 channels.
- There are two types of servo "DSPGATE" Gate-Array ICs: The PMAC(1) type that only allows the control of analog amplifiers with ±10 Volts command signals and the PMAC2 type that is also capable of digital direct PWM or stepper command signals.
- Each PMAC channel provided by a PMAC(1) DSPGATE has one DAC output, one encoder input and four dedicated flag inputs: two end-of-travel limits, one home input and one amplifier fault input.
- Any PMAC can control up to eight motors or axis as long as enough channels are provided. Every PMAC contains one DSPGATE, which has channels 1 through 4 (*Mini*-PMAC has only two channels). If Option 1 is ordered (not available on PMAC-*Lite* or *Mini*-PMAC), a second DSPGATE is provided, which has channels 5 through 8. If Accessory 24 is ordered (not available on PMAC-STD), a third DSPGATE is provided, which has channels 9 through 12. If Accessory 24 Option 1 is ordered as well (not available on PMAC-STD), a fourth DSPGATE is provided, which has channels 13 through 16.
- PMAC has its own memory and microprocessor. Therefore, any version of PMAC may run as a standalone controller or a host computer may command it either over a serial port or over a bus port.

1. 3 - Types of PMAC

1.3.1 - PMAC(1) PC or PMAC(1) VME features

Standard Features

- Motorola DSP 56k Digital Signal Processor
- 4 output digital-to-analog (DAC) converters
- 4 full encoder channels
- 16 General Purpose I/O, OPTO-22 compatible
- Multiplexer port for expanded I/O
- Overtravel limit, home, amplifier fault/enable flags
- Display port for LCD and VFD displays
- Bus and/or RS-422Control
- Stand-Alone Operation
- G-Code Command Processing for CNC
- Linear and Circular Interpolation
- 256 motion programs capacity
- Asynchronous PLC program capability
- Rotating buffer for large programs
- 36-bit position range (+/- 64 billion counts)
- 16-bit DAC Output Resolution
- "S-curve" Acceleration and Deceleration
- Cubic Trajectory Calculations, Splines
- Electronic gearing
- Advanced PID servo motion algorithms

Optional Features

- Up to 16 digital-to-analog (DAC) converters outputs
- Up to 16 full encoder channels
- 8Kx16 Dual-ported RAM
- Flash Memory (no battery)
- 40, 60 or 80 MHz CPU
- Extended (Pole-Placement) Servo Algorithm
- Super-high accuracy clock crystal (<10 ppm)
- Voltage-to-frequency (V/F) converters
- 12-bit resolver-to-digital converter inputs
- Sinusoidal encoder feedback inputs
- Yaskawa absolute encoders inputs
- Analog feedback inputs
- MLDTs feedback inputs
- Parallel binary feedback
- Optically isolated encoder inputs
- RS-232 or RS-422 serial communication converters
- Analog-to-Digital Converted inputs
- On-board Voltage to frequency converter
- Up to a total of 2048 multiplexed I/O points
- Up to 100 meters remote I/O operation

1.3.2 - PMAC PC

Recommended for applications with more than four channel requirements in either a PC based or stand alone environment. More than four channels could be used for more than four motors operation, dual-feedback axis (two encoder input each) or commutated motors (two DACs each). For three or four channels applications the PMAC *Lite* board is suggested instead.

1.3.3 - PMAC *Lite*

Recommended for applications with three or four channel requirements in either a PC based or stand alone environment. The term "*Lite*" stands for the limitation of only one "DSPGATE" Gate-Array IC on board. The number of channels could always be expanded, from 4 to 12, through the use of an ACC-24P. The PMAC *Lite* board is also provided in a stand-alone box, the *PMACPack*, complete of power supplies and connectors. For one or two channels applications the *Mini*-PMAC board is suggested instead.

1.3.4 - PMAC VME

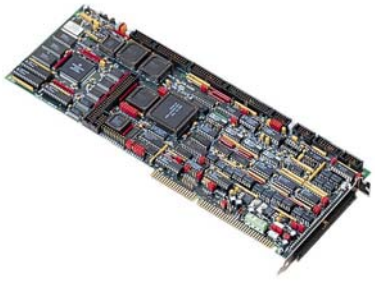
With the same features as the PMAC PC, the PMAC VME is the only option for VME based applications. The PMAC VME could be ordered with either 4 or 8 axes (Option 1). The Dual-ported RAM option in a PMAC VME is on-board.

1.3.5 - PMAC STD

With the same features as the PMAC PC, the PMAC STD is the only option for STD based applications. The Dual-ported RAM option is not available for the PMAC STD and it is limited to eight channels, no ACC-24 is available for it.

PMAC Types

PMAC PC



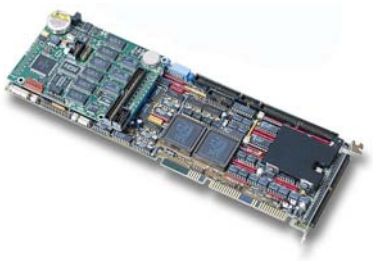
PMAC Lite

Mini PMAC

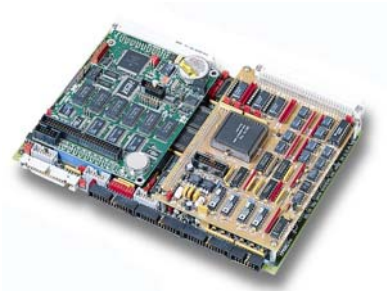
PMAC VME

PMAC STD

PMAC Pack

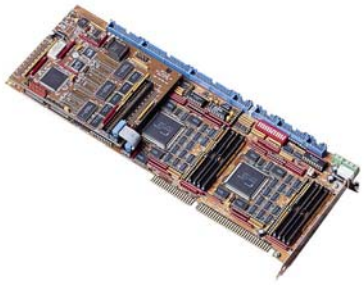


Turbo PMAC PC

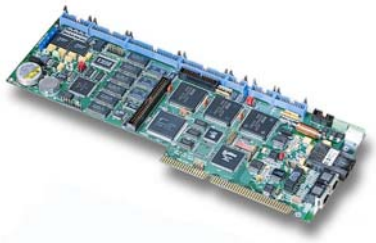


Turbo PMAC VME

Turbo PMAC2 3U



Turbo PMAC2 PC



Turbo PMAC2 PC Ultralite

UMAC Turbo System

Some PMAC types have been omitted from this page. Pictures are not in the same scale.

1.3.6 – *Mini*-PMAC

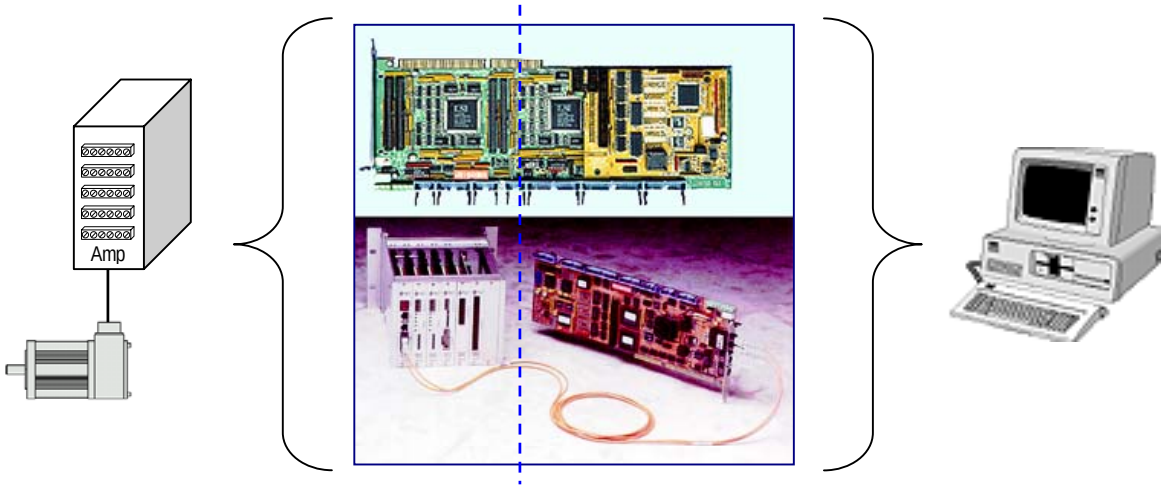
Recommended for applications with one or two channel requirements in either a PC based or stand alone environment. The Dual-ported RAM option in a *Mini*-PMAC is on-board. Two extra full encoder channels (for a total of 4 on-board) could be used for dual feedback applications or, with the two optional voltage-to-frequency (V/F) converters, for stepper drivers or hybrid amplifiers control. There is no control panel port or bus interrupt in the *Mini*-PMAC board. The *Mini*-PMAC board is also provided in a stand-alone box, the *MiniPack*, complete of power supplies and connectors.

1.3.7 – PMAC2

PMAC2 is available in either PC or VME formats. It is suggested for applications that require a digital amplifier control (direct PWM signals) or applications with a combination of analog and digital axis. PMAC2 is also recommended for the use of its built-in features that are optional in PMAC (1): pulse and direction outputs, MLDT inputs, optional 12-bits analog to digital inputs, two extra encoder inputs, improved position compare/capture feature and one channel of parallel feedback. The PMAC2 is available with 4 or 8 axes, with only 4 axes as the PMAC2 *Lite* and with only 2 axes as the *Mini*-PMAC2.

1.3.8 – PMAC2 Ultralite

The term *UltraLite* stands for no "DSPGATE" Gate-Array ICs on board of this kind of PMAC2. The ASICs are located in a different set of boards, usually remotely located from PMAC2, referred as MACRO stations. In fact, the PMAC2 UltraLite in combination with the MACRO station could be seen as a PMAC2 divided in two halves: the central processing portion that contains the DSP processor and the distributed circuitry that connects to motors, amplifiers and different I/O points:



The PMAC2 and the MACRO (Motion And Control Ring Optical) stations are linked with a fiber optic or twisted pair connection. This clever distribution of components brings many benefits: drastic reduction of wiring complexity, elimination of interference by electromagnetic noise and long distance connections (3000 m, ~2 miles with glass fiber).

1.3.9 – Turbo PMAC family

The Turbo PMAC is based in the 56300 Motorola's DSP processor. Its power and speed allows handling up to 32 axes in up to 16 different coordinate systems. In compare with other PMACs, the Turbo PMAC has a highly improved lookahead feature that allows tighter control of acceleration and more accurate cornering profiles.
Motion programs and PLCs developed for other versions of PMAC are compatible with Turbo PMAC. The main difference in the setup is the increased number of variables, necessary to control up to 32 axes. The main Turbo PMAC board has the necessary hardware to connect up to eight channels. The number of channels could be expanded from 8 to 40 by means of either the ACC-24P or ACC-24P2 for PMAC(1) style or PMAC2 respectively. The Turbo PMAC2 is also provided in a 3U format and it is the main component of the UMAC (Universal Motion and Automation Controller) products.

1.4 – PMAC(1) connectors and indicators

1.4.1 – Display Port Outputs (JDISP Port)

The JDISP connector (J1) allows connection of the ACC-12 or ACC-12A liquid crystal displays, or of the ACC-12C vacuum fluorescent display. Both text and variable values may be shown on these displays through the use of the **DISPLAY** command, executing in either motion or PLC programs.

1.4.2 – Control-Panel Port I/O (JPAN Port)

The JPAN connector (J2 on PMAC-PC, -Lite, -VME, and top board of PMAC-STD) is a 26-pin connector with dedicated control inputs, dedicated indicator outputs, a quadrature encoder input, and an analog input. The control inputs are low-true with internal pull-up resistors. They have predefined functions unless the control-panel-disable I-variable (I2) has been set to 1. If this is the case, they may be used as general-purpose inputs by assigning M-variable to their corresponding memory-map locations (bits of Y address $FFC0).

1.4.3 – Thumbwheel Multiplexer Port I/O (JTHW Port)

The Thumbwheel Multiplexer Port, or Multiplexer Port, on the JTHW (J3) connector has eight input lines and eight output lines. The output lines can be used to multiplex large numbers of inputs and outputs on the port, and Delta Tau provides accessory boards and software structures (special M-variable definitions) to capitalize on this feature. Up to 32 of the multiplexed I/O boards may be daisy-chained on the port, in any combination.

1.4.4 – Serial Port Connection

For serial communications, use a serial cable to connect your PC's COM port to the PMAC's serial port connector (J4 on PMAC-PC, -Lite, and -VME; J1 on PMAC-STD's bottom board). Delta Tau provides cables for this purpose: accessory 3D connects PMAC-PC or -VME to a DB-25 connector; ACC-3L connects PMAC-Lite to a DB-9 connector; and ACC-3S connects PMAC-STD to a DB-25 connector. Standard DB-9-to-DB-25 or DB-25-to-DB-9 adapters may be needed for your particular setup.

1.4.5 – General-Purpose Digital Inputs and Outputs (JOPTO Port)

PMAC's JOPTO connector (J5 on PMAC-PC, -Lite, and -VME) provides eight general-purpose digital inputs and eight general-purpose digital outputs. Each input and each output has its own corresponding ground pin in the opposite row. The 34-pin connector was designed for easy interface to OPTO-22 or equivalent optically isolated I/O modules. Delta Tau's Accessory 21F is a six-foot cable for this purpose. The PMAC-STD has a different form of this connector from the other versions of PMAC. Its JOPT connector (J4 on the base board) has 24 I/O, individually selectable in software as inputs or outputs.

1.4.6 – Machine Connectors

The primary machine interface connector is JMACH1 (J8 on PMAC-PC, J11 on PMAC-Lite, P2 on PMAC-VME, J4 on PMAC-STD top board). It contains the pins for four channels of machine I/O: analog outputs, incremental encoder inputs, and associated input and output flags, plus power-supply connections. The next machine interface connector is JMACH2 (J7 on PMAC-PC, P2A on PMAC-VME, J4 on the middle board of an 8-channel PMAC-STD, not available on a PMAC-Lite). It is essentially identical to the JMACH1 connector for one to four more axes. It is only present if the PMAC card has been fully populated to handle eight axes (Option 1), because it interfaces the optional extra components.

1.4.7 – LED Indicators

PMACs with the Option CPU have 3 LED indicators: red, yellow, and green. The red and green LEDs have the same meaning as with the standard CPU: when the green LED is lit, this indicates that power is applied to the +5V input; when the red LED is lit, this indicates that the watchdog timer has tripped and shut down the PMAC.
The new yellow LED located beside the red and green LEDs, when lit, indicates that the phase-locked loop that multiplies the CPU clock frequency from the crystal frequency on the Option CPU is operational and stable. This indicator is for diagnostic purposes only; it may not be present on your board.

1.5 - Working with PMAC

When used for the first time the card must be configured for a specific application, using both hardware and software features, in order to run that application properly. PMAC is shipped from the factory with defaults set in hardware and software set up to be satisfactory for the most common application types. Working with PMAC is very simple and its ease of use and power is based in the following features:

- A clever interrupt-driven scheme allows every task, each motion program and PLC, to run independently of each other.
- Pointer M-variables allow monitoring virtually any register in PMAC's memory from different sources: motion programs, PLCs or the host computer.
- Communications are continuously activated. At any moment, any variable or status command could be interrogated.
- Up to 8 Axes could be either synchronized together, controlled individually or in any combination in between.
- Data gathering and reporting functions allows saving data such as motion trajectories, velocity profiles or any set of variables for later analysis and plot.

1.5.1 - Hardware Setup

On the PMAC, you will see many jumpers (pairs of metal prongs), called E-points (on the bottom board of the PMAC-STD they are called W-points). Some have been shorted together; others have been left open. These jumpers customize the hardware features of the board for a given application. It is strongly recommended to check each jumper configuration using the appropriate hardware reference for the particular PMAC being set. Further instructions for the jumper setup could be found in the PMAC User's manual. After all the jumpers have been properly set PMAC can be installed either inside the host computer or linked with a serial cable to it.

1.5.2 - Software Setup

PMAC has a large set of Initialization parameters (I-variables) that determine the "personality" of the card for a specific application. Many of these are used to configure a motor properly. Once set up, these variables may be stored in non-volatile EAROM memory (using the **SAVE** command) so the card is always configured properly (PMAC loads the EAROM I-variable values into RAM on power-up).
The easiest way to program, setup and troubleshoot PMAC is by using the PMAC Executive Program PEWIN and its related add-on packages P1Setup and PMACPlot. PEWIN has the following main tools and features:

- A terminal window. This is the main channel of communication between the user and PMAC.
- Watch window for real-time system information and debugging.
- Position window for displaying the position, velocity and following error of all motors on the system.
- Several ways to tune PMAC systems.
- Interface for data gathering and plotting.

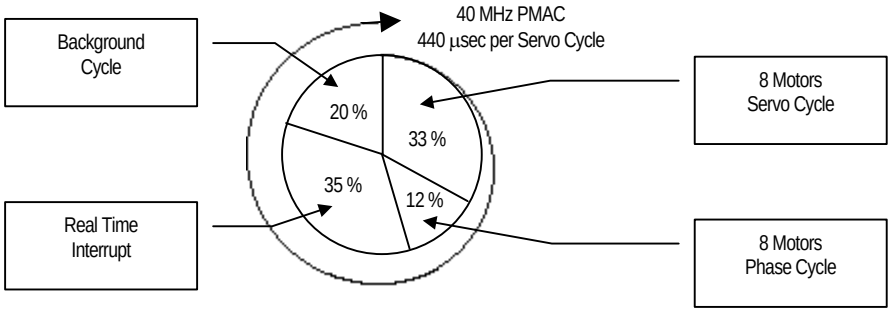
In PEWIN the value of an I- variable may be queried simply by typing in the name of the I- variable. For instance, typing **I900<CR>** causes the value of the I900 to be returned. The value may be changed by typing in the name, an equals sign, and the new value (e.g. **I900=3<CR>**). Remember that if you change any I-variables during this setup, you must use the **SAVE** command before you power down or reset the card, or you will lose the changes that you have made.

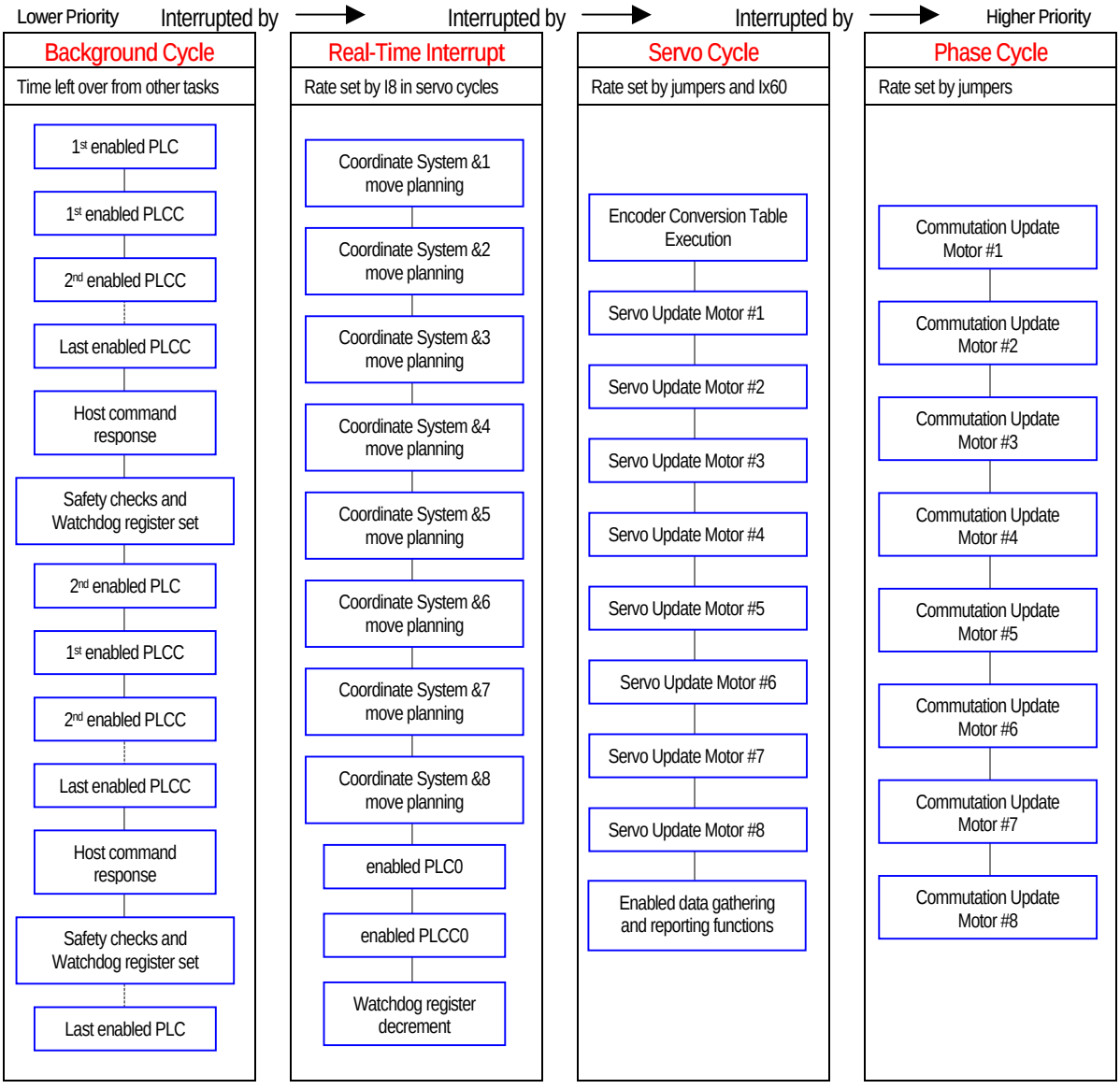
1.5.3 - Programming PMAC

Motion or PLCs programs are entered in any text file and then downloaded with PEWIN to PMAC. PEWIN provides a built-in text editor for this purpose but any other text editor could be conveniently used. Most PMAC commands can be issued from any terminal window communicating with PMAC. Online commands allow, for example, to jog motors, change variables, report variables values, start and stop programs, query for status information and even write short programs and PLCs. In fact, the downloading process is just a sequence of valid PMAC commands sent line by line by PEWIN from a particular text file.

1.6 - PMAC Tasks

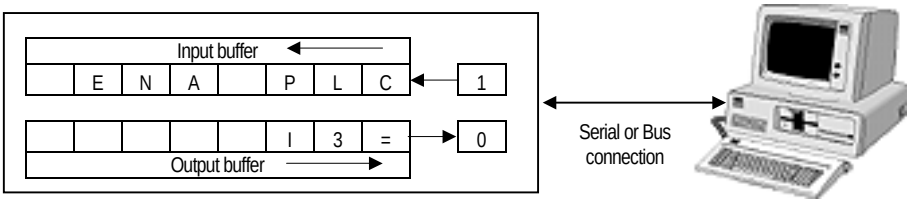
As an example, a 40 MHz PMAC could perform the following tasks with the estimated percentage of the total computational power as indicated:





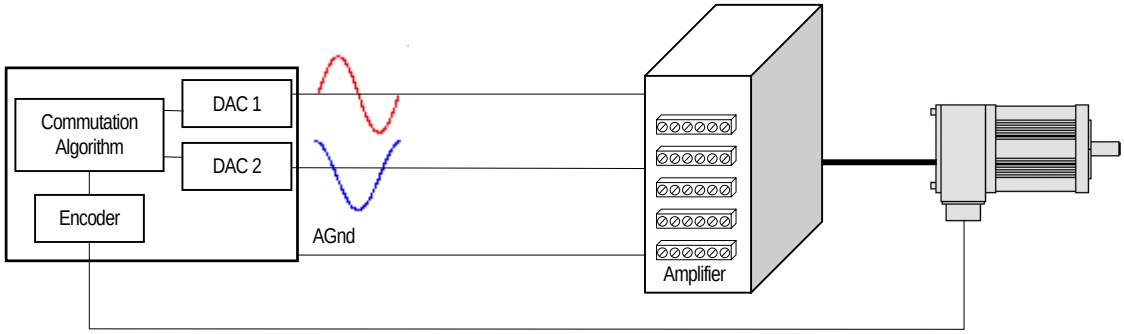
1.6.1 - Single Character I/O

Bringing in a single character from, or sending out a single character to, the serial port or host port (PC or STD) is the highest priority in PMAC. This task takes only 200 nsec per character, but having it at this high priority ensures that the host cannot outrun PMAC on a character-by-character basis. This task is never a significant portion of PMAC's total calculation time. Note that this task does not include processing a full command; that happens at a lower priority (see background tasks below).



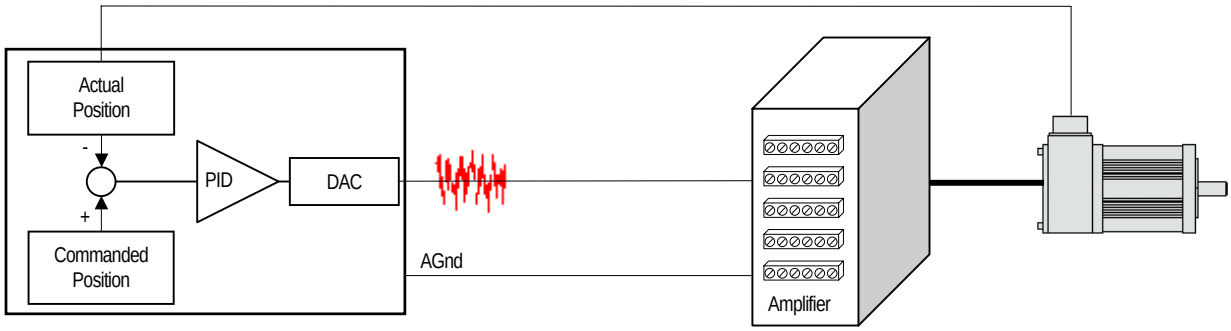
1.6.2 - Commutation Update

The commutation (phasing) update is the second highest priority on PMAC. In a 20 MHz PMAC this task takes 3 µsec per update cycle for each motor commutated by PMAC (Ix01=1). The master clock frequency, and jumpers E98, E29-E33, determines the frequency of this task. The default update frequency is 9 kHz (110 µsec cycle). At the default, the commutation of each motor takes approximately 3% of PMAC's computational power.



1.6.3 - Servo Update

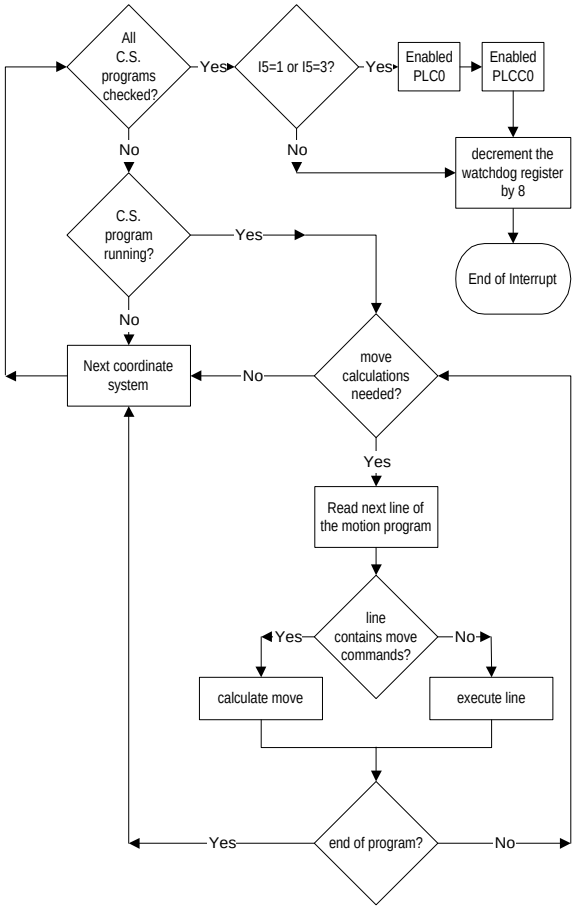
The servo update -- computing the new commanded position, reading the new actual position, and computing a command output based on the difference between the two -- is the third highest priority on PMAC. In a 20 MHz PMAC this task takes 30 µsec per update cycle for each activated motor (Ix00=1) plus about 30 µsec for general servo tasks such as the encoder conversion table. The frequency of this task is determined by the master clock frequency, and jumpers E98, E29-E33, E3-E6. The default update frequency is 2.26 kHz (442 µsec cycle). At the default, the servo update of each motor takes approximately 7% of PMAC's computational power.



1.6.4 -VME Mailbox Processing

Reading or writing a block of up to sixteen characters through the VME mailbox registers is the fourth highest priority in PMAC. The host controls the rate at which this happens. This never takes a significant portion of PMAC's computational power.

1.6.5 - Real-Time Interrupt Tasks



The real-time interrupt (RTI) tasks are the fifth highest priority on PMAC. They occur immediate after the servo update tasks at a rate controlled by parameter I8 (every I8+1 servo update cycles). There are two significant tasks occurring at this priority level: PLC 0 / PLCC0 and motion program move planning.

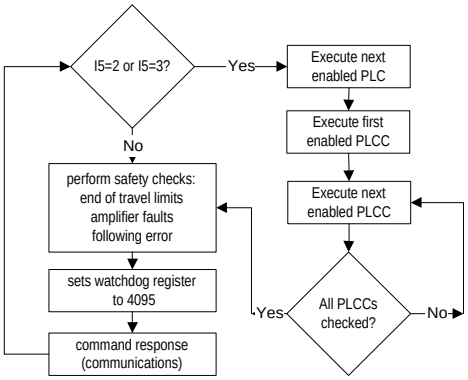
PMAC will scan the lines of each program running in the different coordinate systems and will calculate the necessary number of move commands.

The number of move commands of precalculation could be either zero, one or two and it depends on the type of motion commands and the mode in which the program is being executed.

Non-move commands are executed immediately as they are found. The scan of any given motion program will stop as the necessary number of moves is calculated. It resumes when previous move commands are completed and more move-planning calculations are required.

In the execution of a motion program, if PMAC finds two jumps backward (toward the top) in the program while looking for the next move command, PMAC will pause execution of the program and not try to blend the moves together. It will go on to other tasks and resume execution of the motion program on a later scan. Two statements can cause such a jump back: **ENDWHILE** and **GOTO** (**RETURN** does not count).

1.6.6 - Background Tasks



In the time not taken by any of the higher-priority tasks, PMAC will be executing background tasks. There are three basic background tasks: command processing, PLC programs 1-31, and housekeeping. The frequency of these background tasks is controlled by the computational load on PMAC: the more high-priority tasks are executed, the slower the background tasks will cycle through; and the more background tasks there are the slower they will cycle through.

Each PLC program executes one scan (to the end or to an **ENDWHILE** statement) uninterrupted by any other background task (although it can be interrupted by higher priority tasks). In between each PLC program, PMAC will do its general housekeeping, and respond to a host command, if any.

All enabled PLCC programs execute one scan (to the end or to an **ENDWHILE** statement) starting from lowest numbered to highest uninterrupted by any other background task (although it can be interrupted by higher priority tasks). At power-on\reset PLCC programs run after the first PLC program runs.

The receipt of a control character from any port is a signal to PMAC that it must respond to a command. The most common control character is the carriage return (**<CR>**), which tells PMAC to treat all the preceding alphanumeric characters as a command line. Other control characters have their own meanings, independent of any alphanumeric characters received. Here PMAC will take the appropriate action to the command, or if it is an illegal command, it will report an error to the host.

Between each scan through each background PLC program, PMAC performs its housekeeping duties to keep itself properly updated. The most important of these are the safety limit checks (following error, overtravel limit, fault, watchdog, etc.) Although this happens at a low priority, a minimum frequency is ensured because the watchdog timer will trip, shutting down the card, if this frequency gets too low.

1.6.7 - Observations:

- PMAC has an on-board "watchdog timer" circuit whose job it is to detect a number of conditions that could result in dangerous misfunction. At the default settings, if the RTI frequency were to drop below about 50 Hz or the background cycle is not performed at least every 512 RTI cycles, the timer would trip.
  The purpose of this two-part control of the timer is to make sure all aspects of the PMAC software are being executed, both in foreground (interrupt-driven) and background. If anything keeps either type of routine from executing, the watchdog will fail quickly.

- PLC0 or PLCC0 are meant to be used for only a very few tasks (usually a single task) that must be done at a higher frequency than the other PLC tasks. The PLC 0 will execute every real-time interrupt as long as the tasks from the previous RTI have been completed. *PLC 0 is potentially the most dangerous task on PMAC as far as disturbing the scheduling of tasks is concerned. If it is too long, it will "starve" the background tasks for time.* The first thing you will notice is that communications and background PLC tasks will become sluggish. In the worst case, the watchdog timer will trip, shutting down the card, because the housekeeping task in background did not have the time to keep it updated.

- Although it is very rare for a motion program to cause a watchdog failure, this does happen on occasion. If you have an "empty" (no-motion) loop, the motion program acts much like a PLC 0 during this period. These empty loops, which are usually used to wait for a certain condition, provide very fast response to the change in condition, but their fast repetition occupies a lot of CPU time, and can starve the background tasks for time. Particularly if several coordinate systems are executing empty loops at the same time, you can run into serious background time limitations, which can be severe enough to trip the watchdog timer.
  If there are a huge number of lines of intensive calculations (e.g. 100) before any move or dwell is encountered, there can be such a long time before background calculations are resumed (more than 512 RTI cycles) it is possible to trip the watchdog timer. If this problem occurs, the calculations should be split apart with short DWELL commands to give other tasks time to execute.

- It is possible to use compiled PLCC programs for faster execution. The faster execution of the compiled PLCs comes from two factors: first, from the elimination of interpretation time, and second, from the capability of the compiled PLC programs to execute integer arithmetic.
  The space dedicated to store up to 32 compiled PLC programs, however, is limited to 15K (15,360) 24-bit words of PMAC memory; or 14K (14,336) words if there is a user-written servo as well.

- In between each scan of each individual background interpreted PLC program, PMAC will execute one scan of all active background compiled PLCs. This means that the background compiled PLCs execute at a higher scan rate than the background interpreted PLCs. For example, if there are seven active background interpreted PLCs, each background compiled PLC will execute seven scans for each scan of a background interpreted PLC.

- Most of the housekeeping functions are safety checks such as following error limits and overtravel limits. Since compiled PLCCs are executed at the same rate as the housekeeping functions, code to complement or replace these functions could be placed in a compiled PLCC. If, for example, an extra input flag is desired for position capturing purposes either the end-of-travel limit inputs or the amplifier fault input could be used. The automatic check of the input flag could be disabled by an appropriate setting of the corresponding Ix25 variable and replaced by a PLCC code that will check a general purpose input where the amplifier fault or end-of-travel limit would be connected instead.

- On power-up\reset PLC programs are executed sequentially from 1 to 31. This makes PLC1, the first code executed, the ideal place to perform initialization commands like other PLCs disabling, motors phasing and motion programs start. After its execution PLC1 could disable itself with the command DIS PLC1, running only once on power-up\reset.

- Bits of the first word returned from the global status bits request command, **???** :
  Bit 22 *Real-Time Interrupt Re-entry*: This bit is 1 if a real-time interrupt task has taken long enough so that it was still executing when the next real-time interrupt came (I8+1 servo cycles later). It stays at 1 until the card is reset, or until this bit is manually changed to 0. If motion program calculations cause this it is not a serious problem. If PLC 0 causes this (no motion programs running) it could be serious.
  Bit 20 *Servo Error*: This bit is 1 if PMAC could not properly complete its servo routines. This is a serious error condition. It is 0 if the servo operations have been completing properly.

1.6.8 - Priority Level Optimization

PMAC will usually have enough speed and calculation power to perform all of the tasks asked of it without the user having to worry. Some applications will put a large demand on a certain priority level and to make PMAC run more efficiently some priority level optimization should be done. When PMAC begins to run out of time, problems such as sluggish communications, slow PLC/PLCC scan rates, run-time errors, and even tripping the Watchdog timer can occur.

- The active part of the Encoder Conversion Table is ended by the first Y word that is equal to all zeros. For an application with less than 8-encoders (the default table converts the eight incremental encoder registers on the base PMAC) a last entry with all zeros in the Y word could be defined as necessary.

- Check to see if everything performed in the Real Time Interrupt (RTI) is necessary or if some of it could be moved to a lower priority or slowed down. PLC0 could maybe be done as PLCC1, or the RTI could be done every 4th or 5th servo cycle setting I8=3 or higher.

- Large PLC programs can be split into a few shorter PLC programs. This increases the frequency of Housekeeping and communications by giving more breaks in PLC scans.

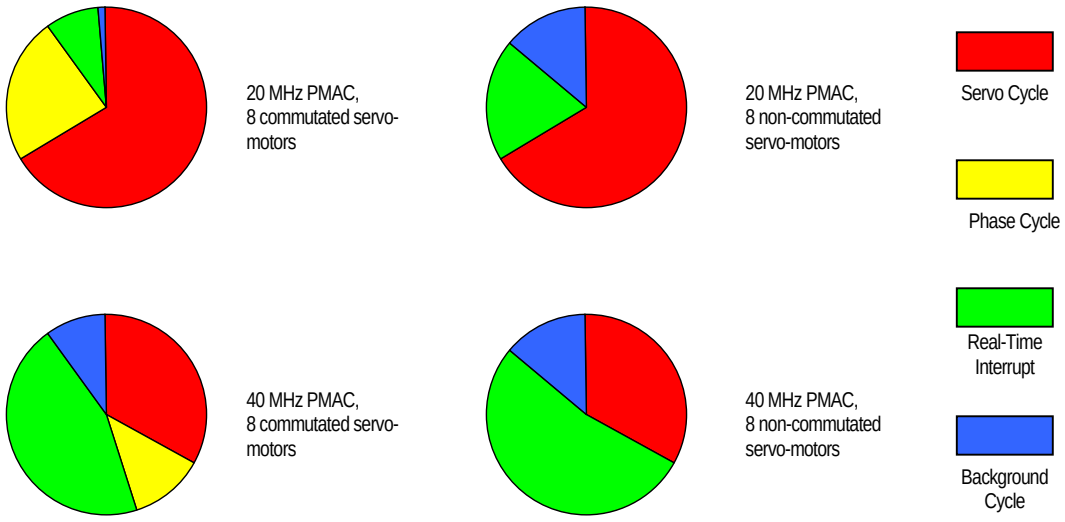
- Motion program **WHILE (condition)WAIT** statements could be done as follows;

  WHILE (condition)
  DWELL20
  ENDWHILE

  This will give more time to other RTI jobs such as Move Planning and PLC/PLCC0.

- If your routines of lower priority than the servo loop are not executing fast enough, you should consider slowing down the servo update rate (increasing the update time). You may well be updating faster than is required for the dynamic performance you need. If so, you are just wasting processor time on needless extra updates. For example, doubling the servo update time from 442 μsec to 885 μsec, virtually doubles the time available for motion and PLC program execution, allowing much faster motion block rates and PLC scan rates. This frequency change could be executed either by jumpers or individually per motor by means of the Ix60 variable.

- A faster than 20 MHz PMAC will perform calculations faster, in proportion to the corresponding clock rate increase. In general a clock rate increase is desirable to increase the real time interrupt (RTI) share of the total computational time available. These cases include applications where large move calculations are involved (small-moves contouring), maintaining the same servo-loop rate and therefore the same control performance.

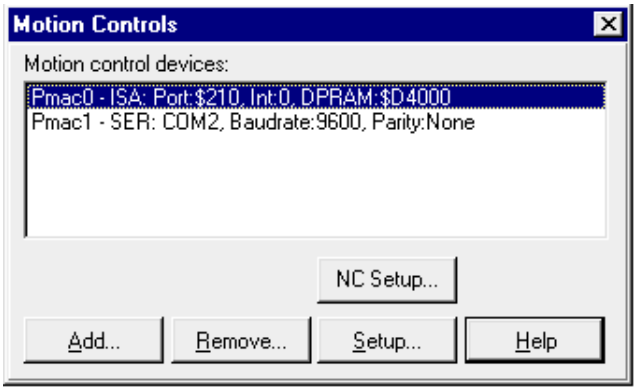


2.0 - PMAC Executive program, PEWIN

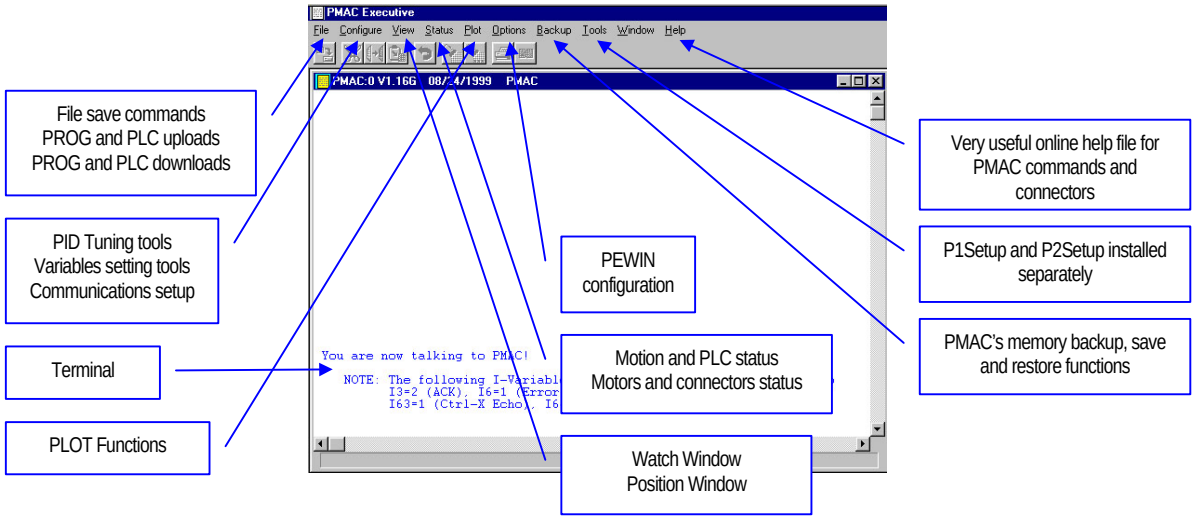
PEWIN enables you to configure, control and trouble-shoot your PMAC (s). PEWIN is designed as a development tool for creating and managing PMAC implementations. It provides a terminal interface to the PMAC and a text editor for writing and editing PMAC motion programs and PLC programs. Additionally, PEWIN contains a suite of tools for configuring and working with PMAC and it's accessories including interfaces for jogging motors, extensive system utilities, screens for viewing various PMAC variables and status registers.

2.1 - Configuring PEWIN

1) Define a new device using the MOTIONEXE.EXE application provided.



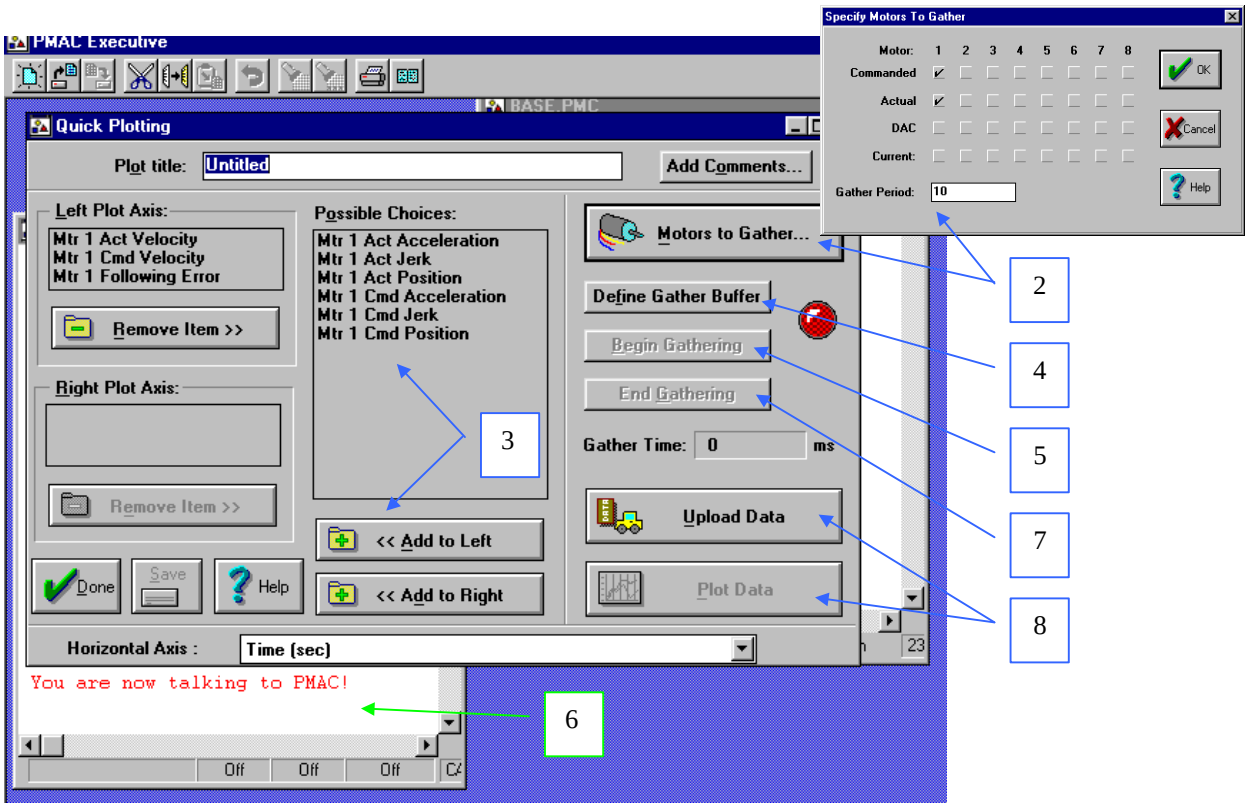
2) Open PEWIN and select the Open Terminal pull-down menu. Select the device created in the previous step.



3) The colors and different options could be set through the Preferences command present in the Options pull-down menu. It is also suggested to disable the automatic status-reporting feature unchecking "Enable Terminal Status Bar" from the Terminal preferences.

2.2 - Quick Plot feature

1) To run the quick plot feature, press ALT+P and press Enter
2) Select the motors to gather and the feature to gather
3) Select what to plot from the possible choices and then press “Add to left” or “Add to right”
4) Press the “Define Gather Buffer” button
5) Press the “Begin Gathering” button
6) Click on the terminal part of the screen and run the motion program or Jog command
7) Press the “End Gathering” button when the motion is completed
8) First press the “Upload Data” button and then the “Plot Data” button.



The Plot feature relies on the PMAC gathering functions. It is very useful for analyzing motion profiles and trajectories. Using circular interpolation, for example, one axis could be plot Vs the other instead of Vs time. Simulating an X-Y plot graphically could be a very important aid in understanding the set of parameters involved in a circular interpolation move.

2.3 - Saving and retrieving PMAC parameters

It is very important to periodically save the complete set of PMAC parameters in the host computer. In case of a failure or replacement, a single file created this way will allow restoring all the variables and programs necessary for the particular application. To activate this function click on the terminal window, press CTRL+B for the Backup pull-down menu, select “save configuration” and “global configuration”. Select a meaningful name to be saved as. Usually is a good practice to include the date as part of the file name for later identification. For example, “ PMAC0112” has four digits for the application identifier and four digits for the date.
After the file is saved verify it with the feature part of the same pull-down menu. This will make sure PMAC’s memory matches the recently saved file and therefore that it is a valid restoring file.
To restore a configuration simply select “restore” from the same Backup pull-down menu. It is also suggested to verify PMAC’s memory after the restore function as well.

2.4 - The WATCH and POSITION windows

The position window is accessed through the "Position" command of the View pull-down menu, ALT+V and P from the terminal window. It is a very convenient way to continuously check PMAC parameters such as position velocity and following error. Right-clicking on this window allows the items selections as well as its format and update period.
The "Watch" window of the same View menu performs a very similar function. Instead of the motion-related parameters allows to constantly display any variable value in PMAC. Right-clicking on this window allows selecting the display format from hexadecimal, decimal and binary reporting values.

2.5 - Uploading and Downloading files

These functions are accessible through the "File" pull-down menu. The uploading function is of great importance. Through it is possible to open a text editor with the contents of the requested PLC, Motion Program, M-variables definitions or values, I variables values and so on. This allows not only to check what commands or values PMAC has actually in memory but also will indent IF conditions and WHILE loops, making the program flow better readable. The File pull-down menu also activates a more interactive and complete editor utility, providing a way (also by the communication functions) to compile PLCs and download files including MACRO names.

2.6 - Using MACRO names and Include Files

PEWIN allows using names meaningful for the user in replacement of the common names for variables and functions that PMAC expects (P, Q, M, I):

Example:

File downloaded

```
#define PUMP P1

OPEN PLC1 CLEAR
      PUMP=1
      DISABLE PLC1
CLOSE
```

Uploaded translated PMAC code

```
OPEN PLC 1 CLEAR
      P1=1
      DISPLC1
CLOSE
```

Make sure the "Support MACRO's/PLCCs" option is checked before downloading. The MACRO must be defined before it could be used. In general, MACRO definitions are at the beginning of the text file. MACROs must be up to 255 valid ASCII characters and cannot have spaces in between (the underscore "_" is suggested in place of a space).
The MACRO definitions, or any PMAC code, could be placed in a separate file and be included with a single line in the text file. The file name must be complete of a full path in order for PEWIN to find it.

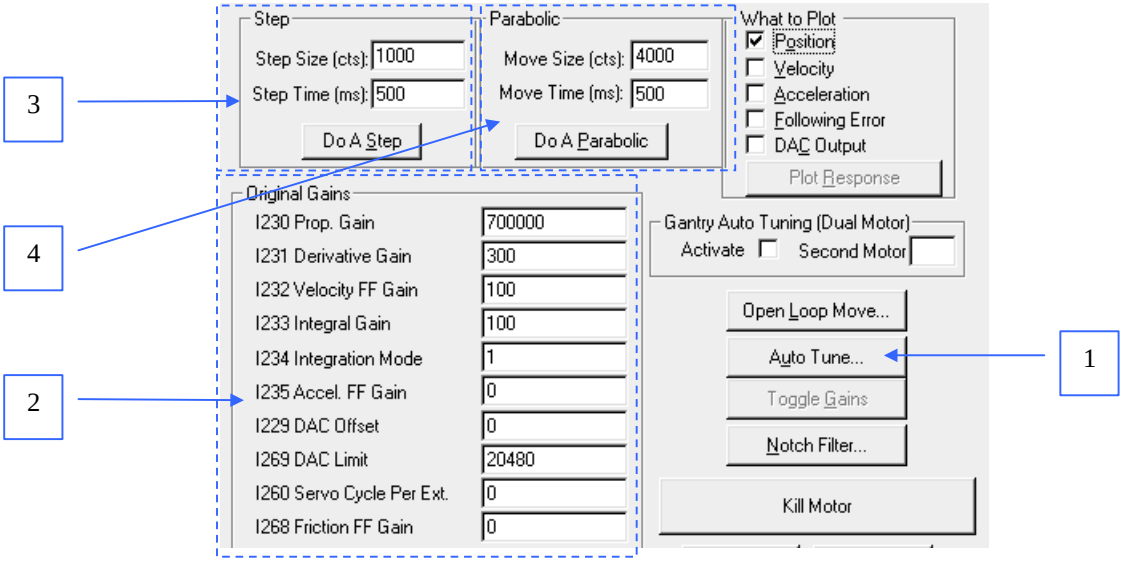
Example: #include "c:\deltatau\files\any.pmc"

2.7 - Downloading compiled PLCCs

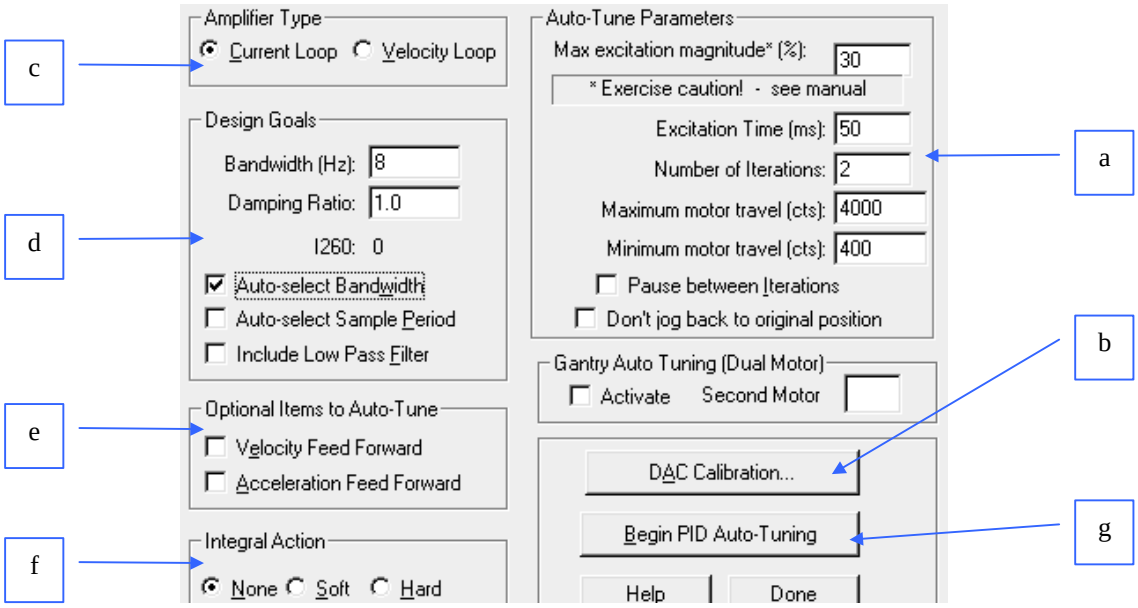
PLCCs are compiled by PEWIN in the downloading process. Only the compiled code gets downloaded to PMAC. Therefore, it is suggested to save the ASCII source code in the host computer separately since it cannot be retrieved from PMAC. Compiled PLCs are firmware dependent and so they must be recompiled when the firmware is changed in PMAC. If more than one PLCC is programmed, all the PLCCs code must belong to the same ASCII text file. PEWIN will compile all the PLCC code present on the file and place it in the appropriate buffer in PMAC. If a single PLCC code is downloaded, all the rest PLCCs that might have been present in memory will be erased, remaining only the last compiled code.
The multiple-file download feature of the PEWIN File menu allows having the PLCC codes in different files. They will be combined by PEWIN in the downloading process.

2.8 - The PID Tuning Utility

This function is accessible from the terminal window by pressing ALT+C for the Configure pull-down menu and T for tuning. The Autotuning feature allows finding the PID parameters with virtually no-effort from the user. The parameters are, in most cases, very close to optimal and in some cases require further fine-tuning by the user.
In this screen press the Page-Up or Page-Down keys on the keyboard to select the motor number.

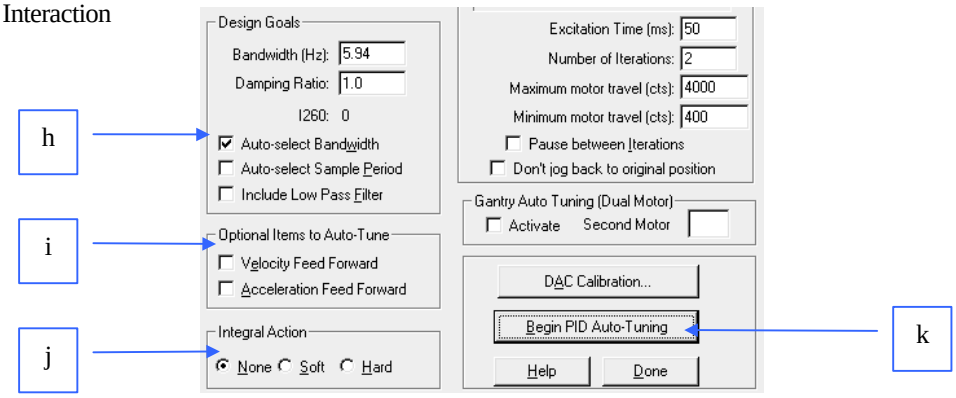


1) Select the Auto Tune feature. This is the first interaction to find a starting bandwidth parameter.



a) Make sure you read the PEWIN manual section related to the safety issues of this procedure.
b) Perform a DAC calibration if necessary.
c) Select the type of amplifier you are trying to tune.
d) Let the Auto Tune select the bandwidth by checking “Auto Select bandwidth”
e) Do not activate any feed forward parameters in this first pass.
f) Do not activate the integral action component in this first pass.
g) Start the first Auto Tuning interaction. Most likely the motor WILL move after Begin is pressed.

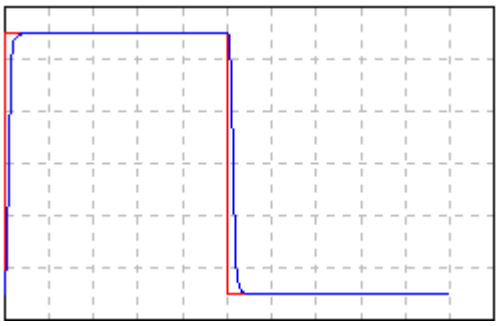
Second Interaction



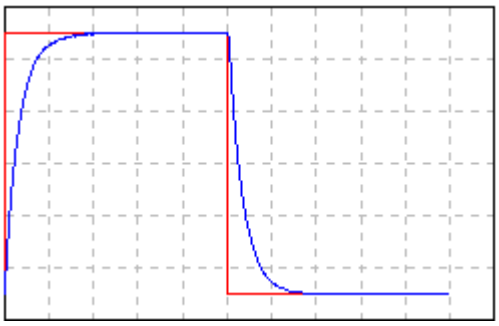
h) The calculated bandwidth could be increased up to three times. Uncheck the "Auto Select bandwidth" this time.
i) Add the feed forward parameters as necessary.
j) Add the integral actions function as necessary.
k) Perform the second pass of the Auto Tuning. After completed remember to select "Implement Now" to activate the selected parameters.

2) After the Auto Tuning is completed the PID parameters can be changed for a final fine tuning approach if necessary.

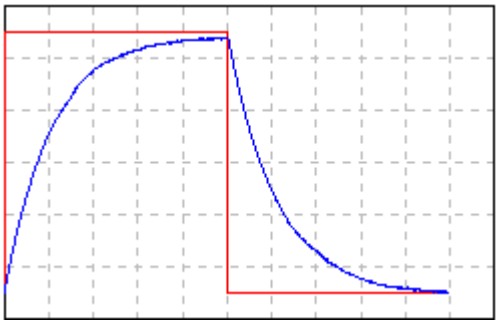
3) Perform a step response and use the following guidelines for the selection of the appropriate I-variables:



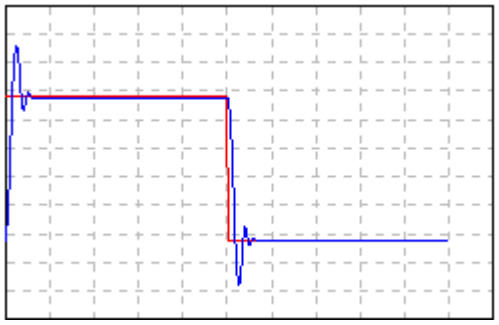
Ideal Case
The motor closely follows the commanded position



Position Offset
Cause: friction or constant force / system limitation
Fix: Increase K_I (Ix33) and maybe use more K_P(Ix30)

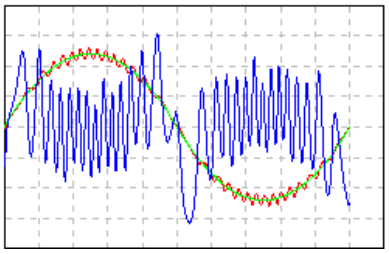


Sluggish Response
Cause: Too much damping or too little proportional gain
Fix: Increase K_P (Ix30) or decrease K_D (Ix31)

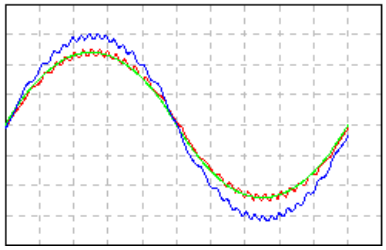


Overshoot and Oscillation
Cause: Too little damping or too much proportional gain
Fix: Decrease K_P (Ix30) or increase K_D (Ix31)

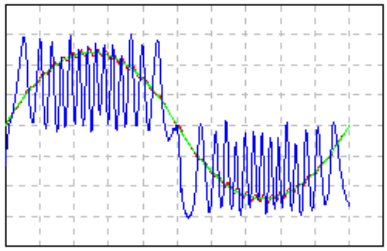
4) Perform a parabolic move and use the following guidelines for the selection of the appropriate I-variables:



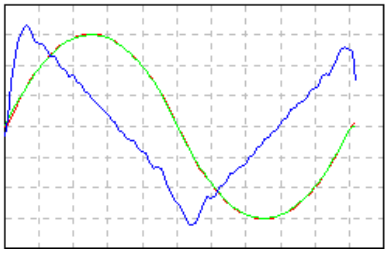
Ideal Case
The following error is reduced at minimum and is concentrated in the center, evenly along the move



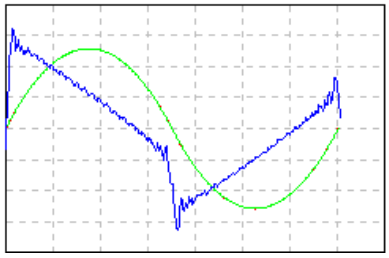
High vel \ FE correlation
Cause: damping
Fix: Increase K_{vel} (Ix32)



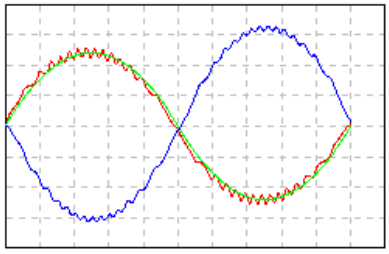
High vel \ FE correlation
Cause: friction
Fix: Increase Integral gain (Ix33) or Friction Feedforward (Ix68)



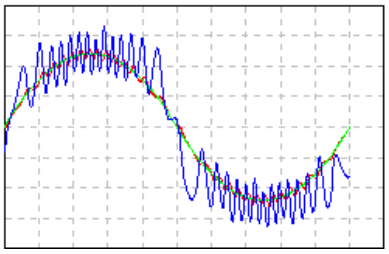
High acc \ FE correlation
Cause: Integral lag
Fix: Increase K_{aff} (Ix35)



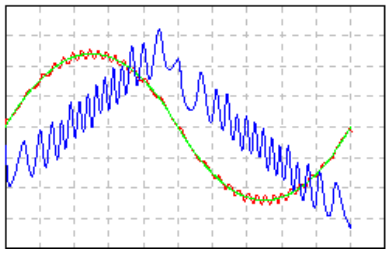
High acc \ FE correlation
Cause: Physical system limitations
Fix: Use less sudden acceleration



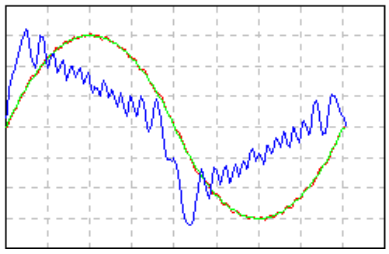
Negative vel \ FE correlation
Cause: Too much velocity FF
Fix: Decrease K_{vel} (Ix32)



High vel \ FE correlation
Cause: damping and friction
Fix: Increase K_{vel} (Ix32)



High acc \ FE correlation
Cause: Too much acc FF
Fix: Decrease K_{aff} (Ix35)



High vel\FE and acc\FEcorrelation
Cause: Integral lag and friction
Fix: Increase K_{aff} (Ix35)

2.9 - Other Features

- Setup of the PMAC encoder conversion table.
- Setup of the Notch and Low Pass Filter parameters.
- Coordinate systems configurations.
- Access to P1Setup and P2Setup (packages provided separately). These setup utilities provide a user-friendly approach for setting up and tuning PMAC (1), with P1Setup, or PMAC2 using P2Setup.
- Online PMAC Software and Hardware help files.
- Jog Ribbon and connector status
- Screens to display, organize or change I, P, Q and M variables.
- Firmware downloading (through MOTIONEXE) for PMACs with flash memory.

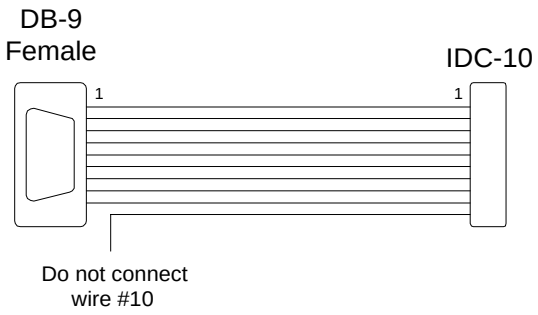
3.0 - Installing and Configuring PMAC

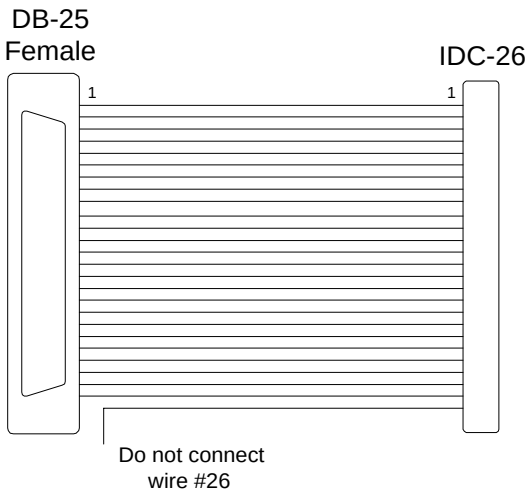
3.1 - Jumpers Setup

On the PMAC, you will see many jumpers (pairs of metal prongs), called E-points (on the bottom board of the PMAC-STD they are called W-points). Some have been shorted together; others have been left open. These jumpers customize the hardware features of the board for a given application. It is strongly recommended to check each jumper configuration using the appropriate hardware reference for the particular PMAC being set. Further instructions for the jumper setup could be found in the PMAC User's manual. After all the jumpers have been properly set PMAC can be installed either inside the host computer or linked with a serial cable to it.

3.2 - Serial Connections

For serial communications, use a serial cable to connect your PC's COM port to the PMAC's serial port connector (J4 on PMAC-PC, -Lite, and -VME; J1 on PMAC-STD's bottom board). Delta Tau provides cables for this purpose: Accessory 3D connects PMAC-PC or -VME to a DB-25 connector; ACC-3L connects PMAC-Lite to a DB-9 connector; and ACC-3S connects PMAC-STD to a DB-25 connector. Standard DB-9-to-DB-25 or DB-25-to-DB-9 adapters may be needed for your particular setup.
If you are using the ACC-26 Serial Communications converter, you will connect from the PC COM port to ACC-26 with a standard DB-9 or DB-25 cable, and from ACC-26 to PMAC using the cable provided with ACC-26. Since the serial ports on PMAC-PC and PMAC-VME are RS-422, this accessory can be useful to provide the level conversion between RS-232 and RS-422 (communications is possible without this conversion, but at reduced noise margin). Because the conversion is optically isolated, the accessory also helps prevent noise and ground-loop problems.
If a cable needs to be made, the easiest approach is to use a flat cable prepared with flat-cable type connectors as indicated in the following diagrams:





3.3 - Establishing Host Communications

Either the Executive or Setup program can be used to establish initial communications with the card. Both programs have menus that allow you to tell the PC where to expect to find the PMAC and how to communicate with it at that location. If you tell it to look for PMAC on the bus, you must also tell it PMAC's base address on the bus (this was set up with jumpers on PMAC). If you tell it to look for PMAC on a COM port, you must tell it the baud rate (this was set up with jumpers or switches on the PMAC). Once you have told the program where and how to communicate with PMAC, it will attempt to find PMAC at that address by sending a query command and waiting for the response. If it gets the expected type of response, it will report that it has found PMAC, and you will be able to proceed. If it does not get the expected type of response after several attempts, it will report that it has not found PMAC.

3.3.1 - Terminal Mode Communications

Once the program reports that it has found PMAC the program should be in terminal emulation mode, so that the PC is acting as a dumb terminal to PMAC. Check to see if you get a response by typing **I10<CR>** (**<CR>** means carriage return, the ENTER or RETURN key). PMAC should respond with a six or seven digit number. If you are not getting the expected results, check the following:

1) Is the green LED (power indicator) on PMAC's CPU board ON, as it should be? If it is not, find out why PMAC is not getting a +5V voltage supply.
2) Is the red LED (watchdog timer indicator) on PMAC's CPU board OFF, as it should be? If it is ON, make sure PMAC is getting very close to 5V supply -- at less than 4.75V, the watchdog timer will trip, shutting down the card. The voltage can be probed at pins 1 and 3 of the J8 connector (A1 and A2 on the PMAC-VME). If the voltage is satisfactory, follow these steps:
   - Turn-off PMAC or the Host computer where it is plugged into.
   - Place the Jumper E51 (the hardware re-initialization jumper) and turn PMAC back on.
   - If PMAC is in bootstrap mode, send a **<CONTROL-R>** character to PMAC to bypass the firmware download.
   - If communications are successful type "$$$***" and "SAVE" in the terminal window.
   - Turn-off PMAC, remove the jumper E51 and try communications again.

Bus Communications

3) Do the bus address jumpers (E91-E92, E66-E71) set an address that matches the bus address that the Executive program is trying to communicate with?
4) Is there something else on the bus at the same address? Try changing the bus address to see if communications can be established at a new address. Address 768 (300 hex) is usually open.

Serial communications

5) Are you using the proper port on the PC? Make sure if the Executive program is addressing the COM1 port, which you have cabled out of the COM1 connector.
6) Does the baud rate specified in the Executive program match the baud rate setting of the E44-E47 jumpers on PMAC?
7) With a breakout box or oscilloscope, make sure you see action on the transmit lines from the PC as you type into the Executive program. If you do not, there is a problem on the PC end.
8) Probe the return communication line while you give PMAC a command that requires a response (e.g. **<CONTROL-F>**). If there is no action, you may have to change jumpers E9-E16 on PMAC to exchange the send and receive lines. If there is action, but the host program does not receive characters, you may have RS-232 receiving circuitry that does not respond at all to PMAC's RS-422 levels. If you have another model of PC, try using it as a test (most models accept RS-422 levels quite well). If you cannot get your computer to accept the signals, you may need a level-conversion device, such as Delta Tau's Accessory-26.

3.3.2 - Resetting PMAC for first time use

Once communications have been established, type the following commands in the terminal window:

```
$$$***                      ;Global Reset
P0..1023=0                  ;Reset P-variables values
Q0..1023=0                  ;Reset Q-variables values
M0..1023->* M0..1023=0      ;Reset M-variables definitions and values
UNDEFINE ALL                ;Undefine Coordinate Systems
SAVE                        ;Save this initial, "clean"configuration
```

3.4 - Connections

Typically, the user connections are actually made to a terminal block that is attached to the JMACH connector by a flat cable (Accessory 8D or 8P). The pinout numbers on the terminal block are the same as those on the JMACH connector for PMAC-PC. While the numbering scheme for the pins on machine connectors on PMAC-VME is different from that for PMAC-PC, the physical arrangement is the same, and PMAC-VME users can use the same terminal numbers on the terminal block board in following the instructions given below.
Make sure PMAC is unpowered while the connections are being made. Leave any loads disconnected from the motor at this point.

3.5 - Power Supplies

3.5.1 - Digital Power Supply

```
1.5A @ +5V (+/-5%) (7.5W)
(Eight-channel configuration, with a typical load of encoders)
```

- The host computer provides the 5 Volts power supply in the case PMAC is installed in its internal bus. With the board plugged into the bus, it will automatically pull +5V power from the bus and it cannot be disconnected. In this case, there must be no external +5V supply, or the two supplies will "fight" each other, possibly causing damage. This voltage could be measured between pins 1 and 3 of the terminal block.
- In a stand-alone configuration, when PMAC is not plugged in a computer bus, it will need an external five-volt supply to power its digital circuits. The +5V line from the supply should be connected to pin 1 or 2 of the JMACH connector (usually through the terminal block), and the digital ground to pin 3 or 4.

3.5.2 - Analog Power Supply

```
0.3A @ +12 to +15V (4.5W)
0.25A @ -12 to -15V (3.8W)
(Eight-channel configuration)
```

The analog output circuitry on PMAC is optically isolated from the digital computation circuitry, and so requires a separate power supply. This is brought in on the JMACH connector. The positive supply -- +12 to +15 volts -- should be brought in on the A+15V line on pin 59. The negative supply -- -12 to -15V -- should be brought in on the A-15V line on pin 60. The analog common (important!) should be brought in on the AGND line on pin 58.
Typically this supply can come from the servo amplifier; many commercial amplifiers provide such a supply. If this is not the case, an external supply may be used. Even with an external supply, the AGND line should be tied to the amplifier common. It is possible to get the power for the analog circuits from the bus, but doing so defeats optical isolation. In this case, no new connections need to be made. However, you should be sure jumpers E85, E87, E88, E89, and E90 are set up for this circumstance. (The card is not shipped from the factory in this configuration.)

3.5.3 - Flags Power Supply (optional)

Each channel of PMAC has four dedicated digital inputs on the machine connector: +LIMn, -LIMn (overtravel limits), HMFLn (home flag), and FAULTn (amplifier fault). In most PMACs these inputs can be kept isolated from other circuits. A power supply from 12 to 24 Volts could be used to power the corresponding opto-isolators related to these inputs. This feature is not available in the PMAC-PC without Option 1, PMAC-VME or the PMAC-STD board.

3.6 - Overtravel limits and Home switches

When assigned for the dedicated uses, these signals provide important safety and accuracy functions. +LIMn and -LIMn are direction-sensitive overtravel limits, that must be actively held low (sourcing current from the pins to ground) to permit motion in their direction. **The direction sense of +LIMn and -LIMn is as follows: +LIMn should be placed at the *negative* end of travel, and -LIMn should be placed at the *positive* end of travel.**

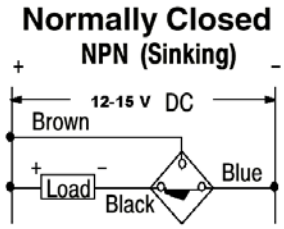
3.6.1 - Disabling the overtravel limits flags

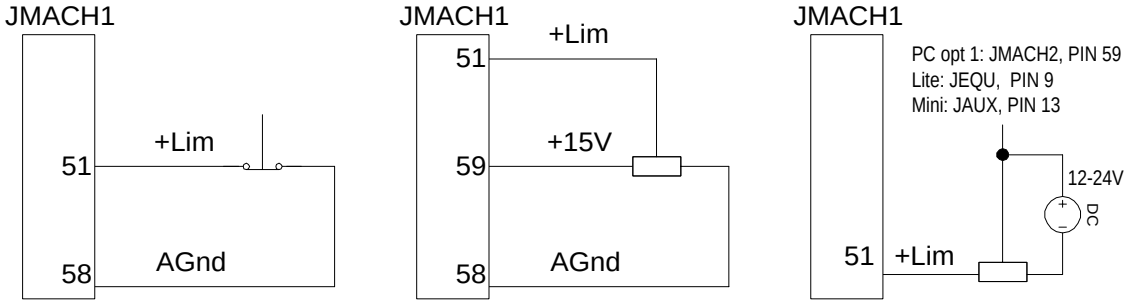
If no overtravel limits are intended to be used, they must be disabled through a change of the variable Ix25. On the terminal window the following commands will disable the limits functions for all 8 motors. Select the motor numbers as appropriate. The OR (|) bit-by-bit function used here is accessible by pressing shift + "\" in the computer's keyboard.

```
I125=I125|$20000 ;Motor #1
I225=I225|$20000 ;Motor #2
I325=I325|$20000 ;Motor #3
I425=I425|$20000 ;Motor #4
I525=I525|$20000 ;Motor #5
I625=I625|$20000 ;Motor #6
I725=I725|$20000 ;Motor #7
I825=I825|$20000 ;Motor #8
```

3.6.2 - Types of overtravel limits

PMAC expects a closed-to-ground connection for the limits to not be considered on fault. This arrangement provides a failsafe condition and therefore it cannot be reconfigured differently in PMAC. Usually a passive normally close switch is used. If a proximity switch is needed instead, the following type is recommended:





Dry Contact | 12-15 Volts proximity | 15-24 Volts proximity

Related PMAC Jumpers must be configured appropriately, following the corresponding PMAC Hardware Reference.

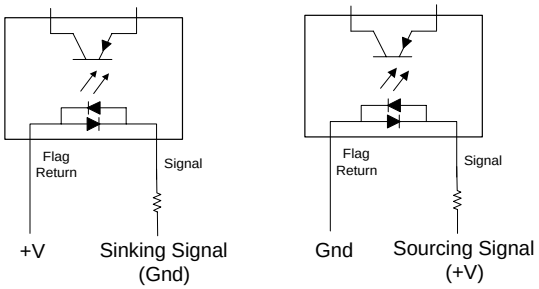
3.6.3 - Home switches

While normally closed-to-ground switches are required for the overtravel limits inputs, the home switches could be either normally close or normally open types. The polarity is determined by the home sequence setup, through the I-variables I902, I907, ... I977. However, for the following reasons, the same type of switches used for overtravel limits are recommended:

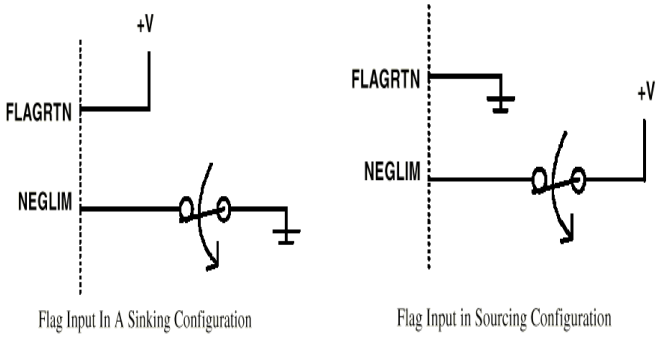
- Normally closed switches are proven to have greater electrical noise rejection than normally open types.
- Using the same type of switches for every input flag simplifies maintenance stock and replacements.

3.6.4 - PMACPack and PMAC2 flag inputs

The PMACPack and PMAC2 interface accessories include a bipolar opto-isolating circuitry (chip PS-2705-4NEC) for flag and amplifier fault connections:



Examples:



3.6.5 - Checking the flag inputs

In the PEWIN terminal window, define the following M-variables for the flags of the motors under consideration:

Flag Type	Motor #1	Motor #2	Motor #3	Motor #4
HMFL input status	M120->X:$C000,20,1	M220->X:$C004,20,1	M320->X:$C008,20,1	M420->X:$C00C,20,1
-LIM input status	M121->X:$C000,21,1	M221->X:$C004,21,1	M321->X:$C008,21,1	M421->X:$C00C,21,1
+LIM input status	M122->X:$C000,22,1	M222->X:$C004,22,1	M322->X:$C008,22,1	M422->X:$C00C,22,1
Flag Type	Motor #5	Motor #6	Motor #7	Motor #8
HMFL input status	M520->X:$C010,20,1	M620->X:$C014,20,1	M720->X:$C018,20,1	M820->X:$C01C,20,1
-LIM input status	M521->X:$C010,21,1	M621->X:$C014,21,1	M721->X:$C018,21,1	M821->X:$C01C,21,1
+LIM input status	M522->X:$C010,22,1	M622->X:$C014,22,1	M722->X:$C018,22,1	M822->X:$C01C,22,1

Open a Watch Window and press insert to enter the M-variable number to watch. Interacting with the switch or sensor monitor a change in the corresponding M-variable. A value of zero indicates that the flag is closed to ground and therefore the limit is not in fault, the motor will be able to run in that direction (*see* Ix25). If the value is 1, the flag is open instead.

3.7 - Motor signals connections

3.7.1 - Incremental Encoder Connection

Each JMACH connector provides two +5V outputs and two logic grounds for powering encoders and other devices. The +5V outputs are on pins 1 and 2; the grounds are on pins 3 and 4. The encoder signal pins are grouped by number: all those numbered 1 (CHA1, CHA1/, CHB1, CHC1, etc.) belong to encoder #1. *The encoder number does not have to match the motor number, but usually does.* If you do not have your PMAC plugged into a bus and drawing its +5V and GND from the bus, use these pins to bring in +5V and GND from your power supply.

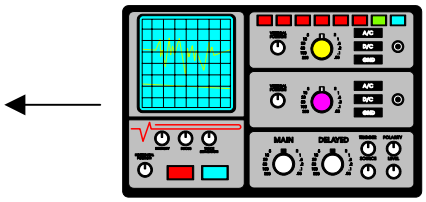
Connect the A and B (quadrature) encoder channels to the appropriate terminal block pins. For encoder 1, the CHA1 is pin 25, CHB1 is pin 21. *If you have a single-ended signal, leave the complementary signal pins floating -- do not ground them.* However, if single-ended encoders are used, please check the settings of the jumpers E18 to E21 and E24 to E27.
For a differential encoder, connect the complementary signal lines -- CHA1/ is pin 27, and CHB1/ is pin 23. The third channel (index pulse) is optional; for encoder 1, CHC1 is pin 17, and CHC1/ is pin 19.

3.7.2 - Checking the encoder inputs

Once the encoders have been properly wired, it is important to check its functionality and its polarity.
Make sure the motor is unpowered while performing this test.
In the PEWIN, open a "Position" window by pressing Alt+V and P from the terminal window. Rotating the encoder monitor the corresponding position value of the motor in the "Position" window. Make sure that a rotation in the positive direction increments the position values. Also, make sure that the number of counts per revolution of the encoder matches the number read by PMAC when a complete revolution of the motor has been rotated. If necessary, for troubleshooting purposes, place an oscilloscope in the encoder inputs to check the appropriate signals provided by the encoder:

Example for encoder #1:

- Channel A in pin 25 of JMACH1 (ACC-8D or ACC-8P)
- Channel B in pin 21 of JMACH1 (ACC-8D or ACC-8P)
- Ground in pin 3 or 4 of JMACH1 (ACC-8D or ACC-8P)



3.7.3 - Checking the DAC outputs

Before connecting the DAC outputs to the amplifier, it is opportune to check the DAC outputs operation.
Make sure the amplifier is not connected while performing this test.
In the PEWIN terminal window, define the following M-variables for the DACs of the motors under consideration:

	Motor #1	Motor #2	Motor #3	Motor #4
DAC output	M102->Y:$C003,8,16,S	M202->Y:$C002,8,16,S	M302->Y:$C00B,8,16,S	M402->Y:$C00A,8,16,S
	Motor #5	Motor #6	Motor #7	Motor #8
DAC output	M502->Y:$C013,8,16,S	M602->Y:$C012,8,16,S	M702->Y:$C01B,8,16,S	M802->Y:$C01A,8,16,S

Example for DAC #1. Type the following in the terminal window:

```
M102->Y:$C003,8,16,S
I100=0
M102=16383
<measure 5 Volts between pins 43 and 58 of JMACH1, (ACC-8D or ACC-8P)>
M102=-16383
<measure -5 Volts between pins 43 and 58 of JMACH1, (ACC-8D or ACC-8P)>
I100=1
```

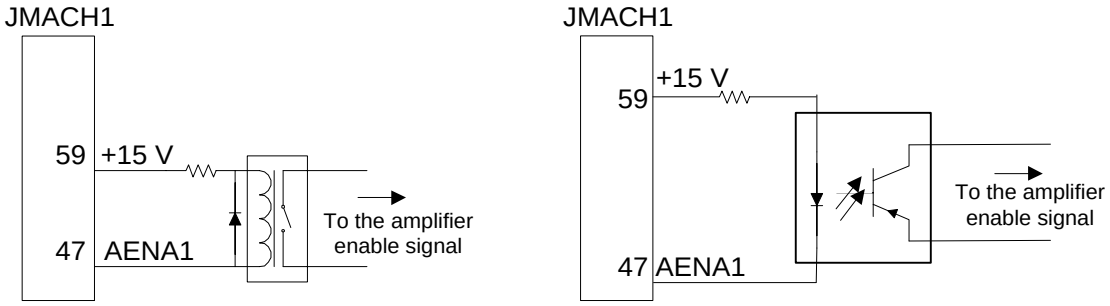
3.7.4 - DAC Output signals

If PMAC is not performing the commutation for the motor, only one analog output channel is required to command the motor. This output channel can be either single-ended or differential, depending on what the amplifier is expecting.
For a single-ended command using PMAC channel 1, connect DAC1 (pin 43) to the command input on the amplifier. Connect the amplifier's command signal return line to PMAC's AGND line (pin 58). *In this setup, leave the* DAC1/ *pin floating; do not ground it.*
For a differential command using PMAC channel 1, connect DAC1 (pin 43) to the plus-command input on the amplifier. Connect DAC1/ (pin 45) to the minus-command input on the amplifier. PMAC's AGND should still be connected to the amplifier common.
If your amplifier is expecting separate sign and magnitude signals, connect DAC1 (pin 43) to the magnitude input. Connect AENA1/DIR1 (pin 47) to the sign (direction input). Amplifier signal returns should be connected to AGND (pin 58). This

format requires some parameter changes on PMAC; (*see* Ix02 and Ix25). Jumper E17 controls the polarity of the direction output; this may have to be changed during the polarity test. This magnitude-and-direction mode is suited for driving servo amplifiers that expect this type of input, and for driving voltage-to-frequency (V/F) converters, such as PMAC's ACC-8D Option 2 board, for running stepper motor drivers.
If you are using PMAC to commutate the motor, you will use two analog output channels for the motor. Each output may be single-ended or differential, just as for the DC motor. The two channels must be consecutively numbered, with the lower-numbered channel having an odd number (e.g. you can use DAC1 and DAC2 for a motor, or DAC3 and DAC4, but not DAC2 and DAC3, or DAC2 and DAC4). For our motor #1 example, connect DAC1 (pin 43) and DAC2 (pin 45) to the analog inputs of the amplifier. If using the complements as well, connect DAC1/ (pin 45) and DAC2/ (pin 46) the minus-command inputs; otherwise leave the complementary signal outputs floating. If you need to limit the range of each signal to +/- 5V, you will do so with parameter I169.

3.7.5 - Amplifier enable signal (AENAx/DIRn)

Most amplifiers have an enable/disable input that permits complete shutdown of the amplifier regardless of the voltage of the command signal. PMAC's AENA line is meant for this purpose. If you are not using a direction and magnitude amplifier or voltage-to-frequency converter, you can use this pin to enable and disable your amplifier (wired to the enable line). AENA1/DIR1 is pin 47. This signal is an open-collector output and requires a pull up resistor to A+15V. For early tests, you may wish to have this amplifier signal under manual control. The polarity of the signal is controlled by jumper (s) E17. The default is low-true (conducting) enable. For any other kind of amplifier enable signal, a dry contact of a relay or a solid-state relay could be used:



The amplifier enable signal could also be manually controlled setting Ix00=0 and using the properly defined Mx14 variable.

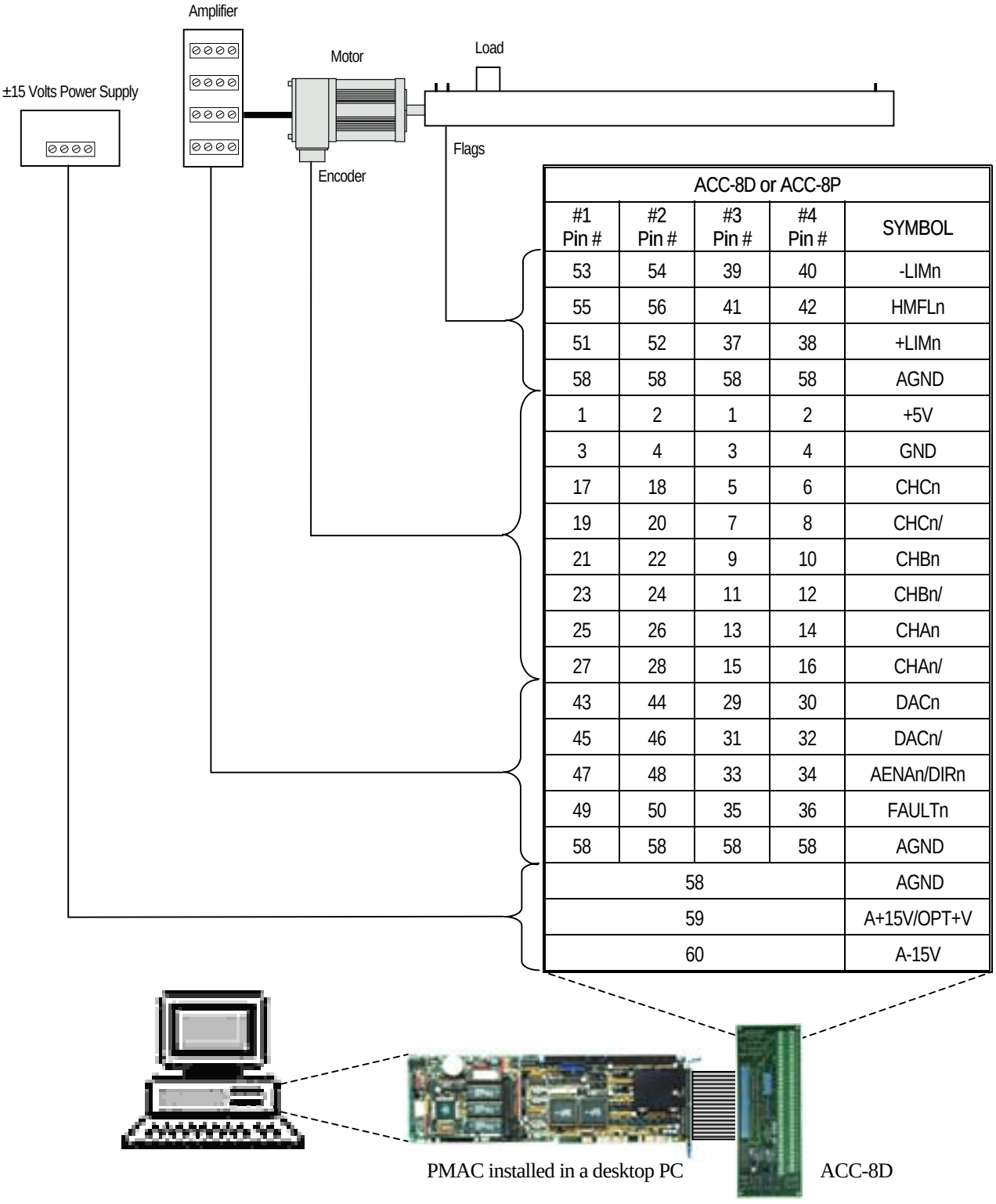
3.7.6 - Amplifier fault signal (FAULTn)

This input can take a signal from the amplifier so PMAC knows when the amplifier is having problems, and can shut down action. The polarity is programmable with I-variable Ix25 (I125 for motor #1) and the return signal is analog ground (AGND). FAULT1 is pin 49. With the default setup, this signal must actively be pulled low for a fault condition. In this setup, if nothing is wired into this input, PMAC will consider the motor not to be in a fault condition.
The amplifier fault signal could be monitored using the properly defined Mx23 variable.

3.7.7 - General-Purpose Digital Inputs and Outputs (JOPTO Port)

PMAC's JOPTO connector (J5 on PMAC-PC, -Lite, and -VME) provides eight general-purpose digital inputs and eight general-purpose digital outputs. Each input and each output has its own corresponding ground pin in the opposite row. The 34-pin connector was designed for easy interface to OPTO-22 or equivalent optically isolated I/O modules. Delta Tau's Accessory 21F is a six-foot cable for this purpose. These inputs and outputs are typically accessed in software through the use of M-variables. In the suggested set of M-variable definitions, variables M1 through M8 are used to access outputs 1 through 8, respectively, and M11 through M18 to access inputs 1 through 8, respectively. This port maps into PMAC's memory space at Y address $FFC2.
The ACC-21S is an I/O simulator for the PMAC JOPTO port; it provides 8 switch inputs and 8 LED outputs.
The ACC-21S is a good tool for I/O simulation and troubleshooting of the JOPTO port in PMAC.

3.8 - Machine Connections Example



ACC-8D or ACC-8P				
#1 Pin #	#2 Pin #	#3 Pin #	#4 Pin #	SYMBOL
53	54	39	40	-LIMn
55	56	41	42	HMFLn
51	52	37	38	+LIMn
58	58	58	58	AGND
1	2	1	2	+5V
3	4	3	4	GND
17	18	5	6	CHCn
19	20	7	8	CHCn/
21	22	9	10	CHBn
23	24	11	12	CHBn/
25	26	13	14	CHAn
27	28	15	16	CHAn/
43	44	29	30	DACn
45	46	31	32	DACn/
47	48	33	34	AENAn/DIRn
49	50	35	36	FAULTn
58	58	58	58	AGND
58				AGND
59				A+15V/OPT+V
60				A-15V

This diagram is just an example of one of the many variations of the machine connections. PMAC Jumpers must be set appropriately following both the appropriate PMAC Hardware Reference and the PMAC User's Manual.

3.9 - Software Setup

PMAC has a large set of Initialization parameters (I-variables) that determine the "personality" of the card for a specific application. Many of these are used to configure a motor properly. Using PEWIN follow these steps for Software Setup:

1) Fully reset PMAC to assure a clean memory configuration before start:

```
$$$***                          ;Global Reset
P0..1023=0   Q0..1023=0         ;Reset P-variables and Q-variables values
M0..1023->* M0..1023=0          ;Reset M-variables definitions and values
UNDEFINE ALL                    ;Undefine Coordinate Systems
SAVE                            ;Save this initial, "clean"configuration
```

2) Define the safety I-variables appropriately (x stands for the motor number, 1 through 8):

	Motor safety I-variables	Range	Default	Units
Ix00	Motor x Activate	0 .. 1	0 (1 for Motor 1)	none
Ix11	Motor x Fatal Following Error Limit	0 .. 8,388,607	32000	1/16 Count
Ix12	Motor x Warning Following Error Limit	0 .. 8,388,607	16000	1/16 Count
Ix13	Motor x + Software Position Limit	+/- 2^{47}	0 (Disabled)	Encoder Counts
Ix14	Motor x - Software Position Limit	+/- 2^{47}	0 (Disabled)	Encoder Counts
Ix15	Motor x Abort/Lim Decel Rate	positive floating point	0.25	Counts/msec^2
Ix16	Motor x Maximum Velocity	positive floating point	32	Counts/msec
Ix17	Motor x Maximum Acceleration	positive floating point	0.015625	Counts/msec^2
Ix19	Motor x Maximum Jog Acceleration	positive floating point	0.015625	Counts/msec^2
Ix25	Motor x Flag Address	PMAC 'X' addresses	see Ix25 table	Extended legal PMAC 'X' addresses

For dual feedback systems: $Ix08 \cdot \frac{\text{Number of counts of the position encoder}}{\text{Units of Distance of the position encoder}} = Ix09 \cdot \frac{\text{Number of counts of the velocity encoder}}{\text{Units of Distance of the velocity encoder}}$

3) **Leave any loads disconnected from the motor at this point.**
Test the polarity and functioning of the motor by means of open loop commands. For the open loop command to work the overtravel limits must be either disabled (*see* Ix25) or properly connected. Type the following in the terminal:

```
#1O10       ; "Pound one, 'O' ten" will output 10% of the DAC on motor #1. It is about 0.6 Volts on default settings
<Observe the motor turning in the positive direction; the position window should indicate motor #1 counting up>
#1O-10      ; "Pound one, 'O' negative ten" will output a negative 10% of the DAC on motor #1, about -0.6 Volts
<Observe the motor turning in the negative direction; the position window should indicate motor #1 decreasing>
```

Slowly increase the percentage of the output command issued if no motion is observed. If after 50% no reaction of the motor occurred, check the DAC outputs following the guidelines in the previous sections.

4) Perform a tuning procedure as described in the PEWIN chapter.

5) After the tuning process have been completed satisfactory, check it by means of the following online commands:

```
SAVE        ;Save this setup
#1J+        ;Jog Motor #1 continuously in the positive direction
#1J-        ;Jog Motor #1 continuously in the negative direction
#1J=2000    ;Jog Motor #1 to a known location
```

6) Create a PMAC's memory backup file as described in the PEWIN chapter.

4.0 - Programming PMAC

Programming PMAC is very simple; the ease of use and power is based in the following features:

- A clever interrupt-driven scheme allows every task, each motion program and PLC, to run independently of each other.
- Pointer M-variables allow monitoring virtually any register in PMAC's memory from different sources: motion programs, PLCs or the host computer.
- Communications are continuously activated. At any moment, any variable or status command could be interrogated.
- Up to 8 Axes could be either synchronized together, controlled individually or in any combination in between.
- Data gathering and reporting functions allows saving data such as motion trajectories, velocity profiles or any set of variables for later analysis and plot.

PMAC is fundamentally a command-driven device, you make PMAC do things by issuing it ASCII command text strings, and PMAC generally provides information to the host in ASCII text strings.
When PMAC receives an alphanumeric text character over one of its ports, it does nothing but place the character in its command queue. It requires a control character (ASCII value 1 to 31) to cause it to take some actual action. The most common control character used is the "carriage return" (**<CR>**; ASCII value 13), which tells PMAC to interpret the preceding set of alphanumeric characters as a command and to take the appropriate action.

4.1 - Online Commands

Many of the commands given to PMAC are on-line commands; that is, they are executed immediately by PMAC, either to cause some action, change some variable, or report some information back to the host.
Some commands, such as **P1=1**, are executed immediately if there is no open program buffer, but are stored in the buffer if one is open. Other commands, such as **X1000 Y1000**, cannot be on-line commands; there must be an open buffer -- even if it is a special buffer for immediate execution. These commands will be rejected by PMAC (reporting an ERR005 if I6 is set to 1 or 3) if there is no buffer open. Still other commands, such as **J+**, are on-line commands only, and cannot be entered into a program buffer (unless in the form of **CMD"J+"**, for instance).

There are three basic classes of on-line commands: motor-specific commands, which affect only the motor that is currently addressed by the host; coordinate-system-specific commands, which affect only the coordinate system that is currently addressed by the host; and global commands, which affect the card regardless of any addressing modes.
A motor is addressed by a **#n** command, where **n** is the number of the motor, with a range of 1 to 8, inclusive. This motor stays the one addressed until another **#n** is received by the card. For instance, the command line **#1J+#2J-** tells Motor 1 to jog in the positive direction, and Motor 2 to jog in the negative direction. There are only a few types of motor-specific commands. These include the jogging commands, a homing command, an open loop command, and requests for motor position, velocity, following error, and status.
A coordinate system is addressed by a **&n** command, where **n** is the number of the coordinate system, with a range of 1 to 8, inclusive. This coordinate system remains the one addressed until another **&n** command is received by the card. For instance, the command line **&1B6R&2B8R** tells Coordinate System 1 to run Motion Program 6 and Coordinate System 2 to run Motion Program 8. There are a variety of types of coordinate-system-specific commands. Axis definition statements act on the addressed coordinate system, because motors are matched to an axis *in a particular coordinate system*. Since it is a coordinate system that runs a motion control program, all program control commands act on the addressed coordinate system. Q-variable assignment and query commands are also coordinate system commands, because the Q-variables themselves belong to a coordinate system.
Some on-line commands do not depend on which motor or coordinate system is addressed. For instance, the command **P1=1** sets the value of P1 to 1 regardless of what is addressed. Among these global on-line commands are the buffer management commands. PMAC has multiple buffers, one of which can be open at a time. When a buffer is open, commands can be entered into the buffer for later execution.
Control character commands (those with ASCII values 0 - 31D) are always global commands. Those that do not require a data response act on all cards on a serial daisy-chain. These characters include carriage return **<CR>**, backspace **<BS>**, and several special-purpose characters. This allows, for instance, commands to be given to several locations on the card in a single line, and have them take effect simultaneously at the **<CR>** at the end of the line (**&1R&2R<CR>** causes both Coordinate Systems 1 and 2 to run).

4.2 - Buffered (Program) Commands

As their name implies, buffered commands are not acted on immediately, but held for later execution. PMAC has many program buffers -- 256 regular motion program buffers, 8 rotary motion program buffers (1 for each coordinate system), and 32 PLC program buffers. Before commands can be entered into a buffer, that buffer must be opened (e.g. **OPEN PROG 3**, **OPEN PLC 7**). Each program command is added onto the end of the list of commands in the open buffer; if you wish to replace the existing buffer, use the **CLEAR** command immediately after opening to erase the existing contents before entering the new ones. After finishing entering the program statements, use the **CLOSE** command to close the opened buffer.

4.3 - COMPUTATIONAL FEATURES

4.3.1 - I-variables

I-Variables (initialization, or setup variables) determines the personality of the card for a given application. They are at fixed locations in memory and have pre-defined meanings. Most are integer values, and their range varies depending on the particular variable. There are 1024 I-variables, from I0 to I1023, and they are organized as follows:

```
  I0 -- I79:   General card setup
 I80 -- I99:   Geared Resolver setup
I100 -- I184:  Motor #1 setup
I185 -- I199:  Coordinate System 1 setup
I200 -- I284:  Motor #2 setup
I285 -- I299:  Coordinate System 2 setup
...
I800 -- I884:  Motor #8 setup
I885 -- I899:  Coordinate System 8 setup
I900 -- I979:  Encoder 1 - 16 setup
I980 -- I1023: Reserved for future use
```

Values assigned to an I-variable may be either a constant or an expression. The commands to do this are on-line (immediate) if no buffer is open when sent, or buffered program commands is a buffer is open.

Examples:

```
I120 = 45
I120 = (I120+P25*3)
```

For I-variables with limited range, an attempt to assign an out-of-range value does not cause an error. The value is automatically "rolled over" to within the range by modulo arithmetic (truncation). For example, I3 has a range of 0 to 3 (4 possible values). The command **I3=5** would actually assign a value of 5 modulo 4 = 1 to the variable.

On PMACs with battery-backed RAM, most of the I-variable values can be stored in a 2K x 8 EEPROM IC with the **SAVE** command. These values are safe here even in the event of a battery-backed RAM failure, so the basic setup of the board is not lost. After a new value is given to one of these I-variables, the **SAVE** command must be issued in order for this value to survive a power-down or reset.

The I-variables that are not saved to EEPROM are held in battery-backed RAM. These variables do not require a **SAVE** command to be held through a power-down or reset, and the previous value is not retained anywhere. These variables are: I19-I44, Ix13, Ix14.

On PMACs with flash memory backup (those with Option 4A, 5A, or 5B), all of the I-variable values can be stored in the flash memory with the **SAVE** command. If there is an EEPROM IC on the board, it is not used. After a new value is given to any I-variable, the **SAVE** command must be issued in order for this value to survive a power-down or reset.

Default values for all I-variables are contained in the manufacturer-supplied firmware. They can be used individually with the **I{constant}=*** command, or in a range with the **I{constant}..{constant}=*** command. Upon board re-initialization by the **$$$*** ** command or by a reset with E51 in the non-default setting, all default settings are copied from the firmware into active memory. The last saved values are not lost; they are just not used.

4.3.2 - P-Variables

P-variables are general-purpose user variables. They are 48-bit floating-point variables at fixed locations in PMAC's memory, but with no pre-defined use. There are 1024 P-variables, from P0 to P1023. A given P-variable means the same thing from any context within the card; all coordinate systems have access to all P- variables (contrast Q-variables, which are coupled to a given coordinate system, below). This allows for useful information passing between different coordinate systems. P-variables can be used in programs for any purpose desired: positions, distances, velocities, times, modes, angles, intermediate calculations, etc.
If a command consisting simply of a constant value is sent to PMAC, PMAC assigns that value to variable P0. For example, if you send the command **342<CR>** to PMAC, it will interpret it as **P0=342<CR>**.
This capability is intended to facilitate simple operator terminal interfaces. It does mean, however, that it is not a good idea to use P0 for other purposes, because it is easy to change this accidentally.

4.3.3 - Q-Variables

Q-variables, like P-variables, are general-purpose user variables: 48-bit floating-point variables at fixed locations in memory, with no pre-defined use. However, the meaning of a given Q-variable (and hence the value contained in it) is dependent on which coordinate system is utilizing it. This allows several coordinate systems to use the same program (for instance, containing the line X(Q1+25) Y(Q2), but to do have different values in their own Q variables (which in this case, means different destination points).
Several Q-variables have special uses that you need to watch for. The **ATAN2** (two-argument arctangent) function automatically uses Q0 as its second argument (the "cosine" argument). The **READ** command places the values it reads following letters A through Z in Q101 to Q126, respectively, and a mask word denoting which variables have been read in Q100. The S ("spindle") statement in a motion program places the value following it into Q127.
Based on that and since a total of 1024 Q-variables are shared between potentially 8 Coordinate Systems (128 variables each), the practical range of the Q-variables to be safely used in motion programs is therefore Q1 to Q99.
The set of Q-variables you are working with in a command depends on the type of command. When you are accessing a Q-variable from an on-line (immediate) command from the host, you are working with the Q-variable for the currently host-addressed coordinate system (with the **&n** command). When you are accessing a Q-variable from a motion program statement, you are working with the Q-variable belonging to the coordinate system running the program. If a different coordinate system runs the *same* motion program, it will use *different* Q-variables.
When you are accessing a Q-variable from a PLC program statement, you are working with the Q-variable for the coordinate system that has been addressed *by that PLC program* with the **ADDRESS** command. Each PLC program can address a particular coordinate system independent of other PLC programs and independent of the host addressing. If no **ADDRESS** command is used in the PLC program, the program uses the Q-variables for C.S. 1.

4.3.4 - M-Variables

To permit easy user access to PMAC's memory and I/O space, M-variables are provided. Generally, a definition only needs to be made once, with an on-line command. On PMACs with battery backup, the definition is held automatically. On PMACs with flash backup, the **SAVE** command must be used to retain the definition through a power-down or reset. The user defines an M- variable by assigning it to a location, and defining the size and format of the value in this location. An M-variable can be a bit, a nibble (4 bits), a byte (8 bits), 1-1/2 bytes (12 bits), a double-byte (16 bits), 2-1/2 bytes (20 bits), a 24-bit word, a 48-bit fixed-point double word, a 48-bit floating-point double word, or special formats for dual-ported RAM and for the thumbwheel multiplexer port.
There are 1024 M- variables (M0 to M1023), and as with other variable types, the number of the M-variable may be specified with either a constant or an expression: M576 or M(P1+20) when read from; the number must be specified by a constant when written to.
The definition of an M-variable is done using the "defines-arrow" (**->**) composed of the minus-sign and greater-than symbols. An M-variable may take one of the following types, as specified by the address prefix in the definition:

```
X:    1 to 24 bits fixed-point in X-memory
Y:    1 to 24 bits fixed-point in Y-memory
D:    48 bits fixed-point across both X- and Y-memory
L:    48 bits floating-point across both X- and Y-memory
DP:   32 bits fixed-point (low 16 bits of X and Y) (for use in dual-ported
      RAM)
```

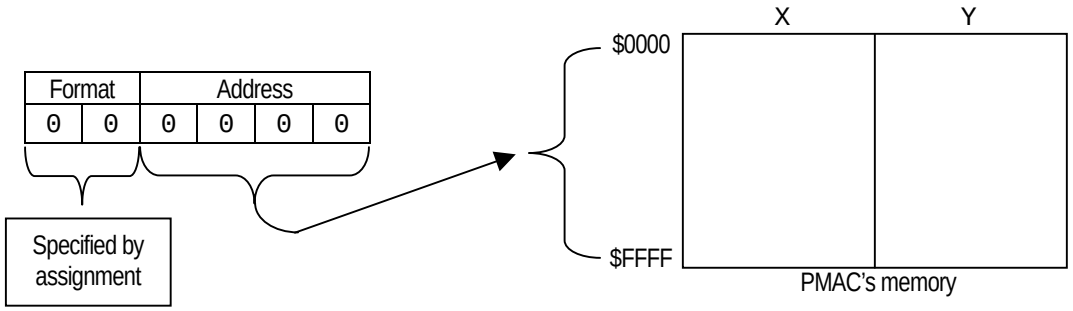
```
F:    32 bits floating-point (low 16 bits of X and Y) (for use in dual-
      ported RAM)
TWD:  Multiplexed BCD decoding from Thumbwheel port
TWB:  Multiplexed binary decoding from Thumbwheel port
TWS:  Multiplexed serial I/O decoding from Thumbwheel port
TWR:  Multiplexed serial resolver decoding from Thumbwheel port
*:    No address definition; uses part of the definition word as general-
      purpose variable
```

If an X or Y type of M-variable is defined, you must also define the starting bit to use, the number of bits, and the format (decoding method).

Typical M-variable definition statements are:

```
M1->Y:$FFC2,8,1
M102->Y:49155,8,16,S
M103->X:$C003,0,24,S
M161->D:$002B
M191->L:$0822
M50->DP:$D201
M51->F:$D7FF
M100->TWD:4,0.8.3,U
```

The M-variable definitions are stored as 24-bit codes at PMAC addresses Y:$BC00 (for M0) to Y:$BFFF (for M1023). For all but the thumbwheel multiplexer port M-variables, the low 16 bits of this code contains the address of the register pointed to by the M-variable (the high 8 bits tell what part of the address is used and how it is interpreted).



If another M-variable points to this part of the definition, it can be used to change the subject register. The main use of this technique is to create arrays of P- and Q-variables or arrays in dual-ported RAM or in user buffers (see on-line command **DEFINE UBUFFER**).

Many M-variables have a more limited range than PMAC's full computational range. If a value outside of the range of an M-variable is attempted to be placed to that M-variable, PMAC automatically "rolls over" the value to within that range and does not report any errors. For example, with a single bit M-variable, any odd number written to the variable ends up as "1", any even number ends up as "0". If a non-integer value is attempted to be placed in an integer M-variable, PMAC automatically rounds to the nearest integer.

Once defined, an M-variable may be used in programs just as any other variable -- through expressions. When the expression is evaluated, PMAC reads the defined memory location, calculates a value based on the defined size and format, and utilizes it in the expression.

Care should be exercised in using M-variables in expressions. If an M-variable is something that can be changed by a servo routine (such as instantaneous commanded position), which operates at a higher priority the background expression evaluation, there is no guarantee that the value will not change in the middle of the evaluation. For instance, if in the expression (M16- M17)*(M16+M17) the M-variables are instantaneous servo variables, the user cannot be sure that M16 or M17 will have the same value both places in the expression, or that the values for M16 and M17 will come from the same servo cycle. The first problem can be overcome by setting P1=M16 and P2=M17 right above this, but there is no general solution to the second problem.

4.3.5 - Array capabilities

It is possible to use a set of P-variables as an array. To read or assign values from the array, simply replace the constant specifying the variable number with an expression in parentheses.

Example:

```
P1=10              ; Array index variable
P3=P(P1)           ; Same as P3=P10
```

To write to the array M-variables must be used. An M-variable defined to the corresponding P-variable address will allow changing any P-variable and therefore the contents of the array.

Example: Values 31 to 40 will be assigned to variables P1 through P10

```
M34->L:$1001       ; Address location of P1
M35->Y:$BC22,0,16  ; Definition word of M34

OPEN PLC 15 CLEAR
P100=31
WHILE (P100!>40)   ; From 31 to 40
    M34=P100       ; Value is written to the array
    P100=P100+1    ; Next value
    M35=M35+1      ; Next Array position (next P-variable)
ENDWHILE
DISABLEPLC15       ; This PLC runs only once
CLOSE

ena PLC15          ; Enable the PLC (I5 must be 2 or 3)
P1..10             ; List the values of P1 to P10
```

The same concept applies for Q-variables and M-variables arrays although the address range for them is different.

4.3.6 - Operators

PMAC operators work like those in any computer language: they combine values to produce new values.
PMAC uses the four standard arithmetic operators: +, -, *, and /. The standard algebraic precedence rules are used: multiply and divide are executed before add and subtract, operations of equal precedence are executed left to right, and operations inside parentheses are executed first.
PMAC also has the '%' modulo operator, which produces the resulting remainder when the value in front of the operator is divided by the value after the operator. Values may be integer or floating point. This operator is particularly useful for dealing with counters and timers that roll over.
When the modulo operation is done by a positive value X, the results can range from 0 to X (not including X itself). When the modulo operation is done by a negative value -X, the results can range from -X to X (not including X itself). This negative modulo operation is very useful when a register can roll over in either direction.
PMAC has three logical operators that do bit-by-bit operations: & (bit-by-bit AND), | (bit-by-bit OR), and ^ (bit-by- bit EXCLUSIVE OR). If floating-point numbers are used, the operation works on the fractional as well as the integer bits. & has the same precedence as * and /; | and ^ have the same precedence as + and -. Use of parentheses can override these default precedences.

4.3.7 - Functions

These perform mathematical operations on constants or expressions to yield new values. The general format is:

{function name} ({expression})

The available functions are **SIN**, **COS**, **TAN**, **ASIN**, **ACOS**, **ATAN**, **ATAN2**, **SQRT**, **LN**, **EXP**, **ABS**, and **INT**.
Whether the units for the trigonometric functions are degrees or radians is controlled by the global I-variable I15.

SIN	This is the standard trigonometric sine function.
COS	This is the standard trigonometric cosine function.

TAN	This is the standard trigonometric tangent function.
ASIN	This is the inverse sine (arc-sine) function with its range reduced to +/-90 degrees.
ACOS	This is the inverse cosine (arc-cosine) function with its range reduced to 0 -- 180 degrees.
ATAN	This is the standard inverse tangent (arc-tangent) function.
ATAN2	This is an expanded arctangent function, which returns the angle whose sine is the expression in parentheses and whose cosine is the value of Q0 for that coordinate system. If doing the calculation in a PLC program, make sure that the proper coordinate system has been ADDRESSed in that PLC program. (Actually, it is only the ratio of the magnitudes of the two values, and their signs, that matter in this function). It is distinguished from the standard ATAN function by the use of two arguments. The advantage of this function is that it has a full 360 degree range, rather than the 180 degree range of the single-argument ATAN function.
LN	This is the natural logarithm function (log base e).
EXP	This is the exponentiation function (e^x). Note: To implement the y^x function, use $e^{x \ln(y)}$ instead. A sample PMAC expression would be EXP(P2*LN(P1)) to implement the function $P1^{P2}$.
SQRT	This is the square root function.
ABS	This is the absolute value function.
INT	This is a truncation function, which returns the greatest integer less than or equal to the argument (INT(2.5)=2, INT(-2.5)=-3).

Functions and operators could be used either in Motion Programs, PLCs or as online commands. For example, the following commands could be typed in a terminal window:

```
P1=SIN (45) P1     ; Reports the sine value of a 45° angle
I130=I130/2        ; Lower the proportional gain of Motor #1 by half
I125=I125|$20000   ; Disable the end-of-travel limits of Motor #1
```

4.3.8 - Comparators

A comparator evaluates the relationship between two values (constants or expressions). It is used to determine the truth of a condition in a motion or PLC program. The valid comparators for PMAC are:

```
=    (equal to)
!=   (not equal to)
>    (greater than)
!>   (not greater than; less than or equal to)
<    (less than)
!<   (not less than; greater than or equal to)
~    (approximately equal to -- within one)
!~   (not approximately equal to -- at least one apart)
```

Note that <= and >= are not valid PMAC comparators. The comparators !> and !<, respectively, should be used in their place.

4.3.9 - User-Written phase and User-Written servo algorithms

For the sophisticated user with unusual and/or difficult commutation needs, PMAC provides the hooks for custom user-written commutation (phasing) or servo algorithms. These routines are to be written in Motorola 56000 assembly language code, usually on a PC or compatible, and cross-assembled for the 56000.

4.4 - Memory Map

PMAC's processor is the Motorola 56001 DSP. The 56001 have dual data buses, each 24 bits wide, so that both operands in a calculation may be brought in simultaneously. Each bus has access to a 16-bit address space (0000hex to FFFFhex), which provides 65,536 24-bit words. One bus and address space is called 'X', and the other is called 'Y'. Therefore, when specifying a single-word memory location, one must use 'X:' or 'Y:' with the 16-bit address. PMAC's input and output is mapped into the same address space with the memory.

PMAC uses double-word memory for both extended fixed-point values and for floating-point values (single words are always fixed point). The fixed-point double-word locations are specified by a 'D:' (double), and the floating-point double-word locations are specified by an 'L:' (long). This matches the syntax of M-variable declarations for these registers.

The user may specify PMAC addresses with either decimal or hexadecimal values; the hex values must be preceded by a '$' to be interpreted as hex. For example, `Y:$FFC0` is the hexadecimal specification, and `Y:65472` is the decimal specification of the same word address.

M-variables are defined by providing the word address, the offset, the width, and the format (irrelevant for bits). Several M-variables were defined at the factory to match to inputs and outputs. For instance, M11 thru M18 were assigned to Machine Inputs 1 thru 8 (MI1-MI8), and M1 to M8 were assigned to Machine Outputs 1 thru 8 (MO1-MO8).

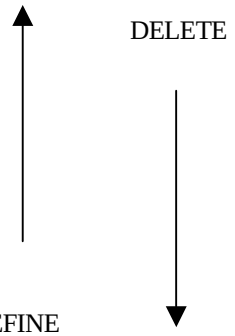
The PMAC architecture is very open, allowing the user to examine and use many internal registers for his own use. Usually this is done through the use of M-variables, which point to locations in the memory-I/O space of the PMAC processor. Once defined to point to the proper location, an M-variable can be treated as any other variable for reading and writing.

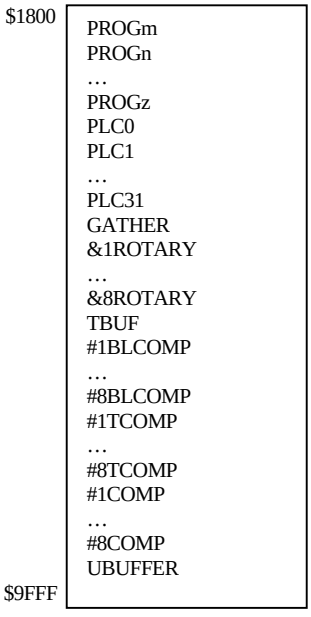
Warning: Certain registers that are under PMAC's automatic control, particularly those used in the servo calculations, can cause problems if the user writes to them directly.

Range	X-Memory	Y-Memory	Type
$0000 - $00FF	Fixed-Use calculation Registers	Fixed-Use calculation Registers	Internal DSP Memory
$0100 - $17FF	Fixed-Use calculation Registers	Fixed-Use calculation Registers	External Static RAM (Battery Backed)
$1800 - $BBFF	User Buffer Storage Space	User Buffer Storage Space	External Static RAM (Battery Backed)
$BC00 - $BFFF	User-Written Servo Storage	M-Variable Definitions	External Static RAM (Battery Backed)
$C000 - $C03F			DSP-Gate Registers
$D000 - $DFFF	Bits 0 to 15	Bits 0 to 15	Dual-Ported RAM
$E000 - $F000	VME Setup Registers (bits 0 to 7)	Mailbox Registers (bits 0 to 7)	VME bus registers
$F000 - $FFFF	N / A		I / O Registers

4.4.1 - User Buffer Storage Space

- 256 Motion Programs can be held. All programs must be stopped before any can be opened.
- All programs must be stopped before any can run.
- A PLC program can be opened while others are running.
- Buffers must be defined from end of memory toward beginning. Buffers must be deleted from beginning of memory to end.



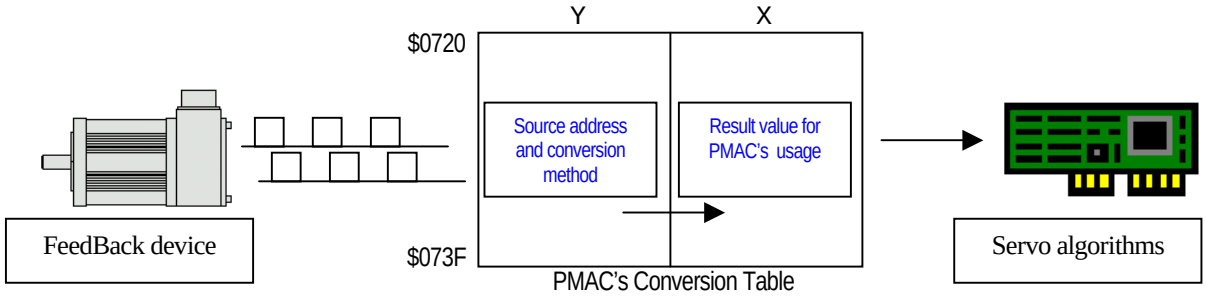


4.5 - Encoder Conversion Table

PMAC uses a multiple-step process to work with its feedback and master position information, and with external time-base sources, to provide maximum power and flexibility. For most PMAC users with quadrature encoders, this process can be virtually transparent, with no need to worry about the details. However, some users will need to understand this conversion process in some detail to make the changes necessary to use other types of feedback, to optimize their system, or to perform special functions. The PMAC Executive Program for PC compatible computers has a special editing screen for the conversion table that makes viewing it and changing it very easy.

4.5.1 - Conversion Table Structure

The Encoder Conversion Table has two "columns", one in the X memory space of the processor, and one in the Y memory space. The X-column holds the converted data, while the Y-column holds the addresses of the source registers, and the conversion methods used on the data in each of those source registers. Basically, the user sets up the table by writing to the Y-column, and PMAC uses the Y-column data to fill up the X-column each servo cycle.



The encoder conversion table starts at address $720 (1824 decimal) in PMAC's memory. It can continue through address $73F (1855 decimal). The active part of the table is ended by the first Y word that is all zeros. The encoder table as shipped from the factory converts the eight incremental encoder registers on the base PMAC board in locations $720 through $727 (1824 to 1831). Locations $728 and $729 create time base information from the converted Encoder 4 register ($723). Y:$72A is zero, ending the active part of the table.
Some conversion types need more than one entry, the other Y-words are further setup parameters for the conversion. The conversion result is placed in the last (highest address) X-word, and the other X-words hold intermediate data.

Example:

$728 (1832)	$400723	Time-base from converted Enc. 4
$729 (1833)	$000295	Time-base scale factor for above

The result of this timebase value based on encoder #4 is placed in register X:$0729, the second and last entry for this conversion.

4.5.2 - Further Position Processing

Once the position feedback signals have been processed by the Encoder Conversion Table (which happens at the beginning of each servo cycle), the data is ready for use by the servo loop. For each activated motor, PMAC takes the position information in the 24-bit register pointed to by Ix03 and extends it in software to a 48-bit register that holds the actual motor position. Several other features are available for conditioning the feedback signal as needed:

- **Axis Position Scaling:** in the coordinate system axis definition a scale factor determines the relationship between encoder counts and user units to be used in motion programs.
- **Leadscrew Compensation:** a compensation table containing corrective values for errors due to the leadscrew imperfections can be created for each motor.
- **Backlash Compensation:** On reversal of the direction of the commanded velocity, a pre-programmed backlash distance is added to or subtracted from the commanded position.
- **Torque Compensation Tables:** The table belonging to a motor provides a torque correction to that motor as a function of that motor's position.

4.6 - PMAC position registers

The PMAC Executive position window or the online "P" command reports the value of the *actual position register* plus the *position bias register* plus the *compensation correction register*, and if bit 16 of Ix05 is 1 (handwheel offset mode), minus the *master position register*:

```
M175->X:$002A,16,1          ; Bit 16 of I105
M162->D:$002B               ; #1 Actual position (1/[Ix08*32] cts)
M164->D:$0813               ; #1 Position bias (1/[Ix08*32] cts)
M167->D:$002D               ; #1 Present master ((handwheel) pos (1/[Ix07*32] cts
                            ; of master or (1/[Ix08*32] cts of slaved motor)
M169->D:$0046               ; #1 Compensation correction
```

$$P100 = \frac{(M162 + M164 + M169 - M175 * M167)}{I108 * 32}$$

P100 will report the same value as the online command "P" or the position window in the PMAC Executive program. The addresses given are for Motor #1. For the registers for another motor x add (x-1)*$3C -- (x-1)*60 -- to the appropriate motor #1 address.)

M161->D:$0028 **; #1 Commanded position (1/[Ix08*32] cts)**

- The motor commanded position registers contain the value in counts where the motor is commanded to move. It is set through **JOG** online commands or axis move commands (**X10**) inside motion programs.

  To read this register in counts: `P161 = M161 / (I108*32)`

M162->D:$002B **; #1 Actual position (1/[Ix08*32] cts)**

- The actual position register contains the information read from the feedback sensor after it has been properly converted through the encoder conversion table and extended from a 24-bits register to a 48-bits register.

  To read this register in counts: `P162 = M162 / (I108*32)`

M163->D:$080B **; #1 Target (end) position (1/[Ix08*32] cts)**

- This register contains the most recent programmed position and it is called the *target position register*. If I13>0 PMAC is in segmentation mode and the value of M163 corresponds to the last interpolated point calculated.

  To read this register in counts: `P163 = M163 / (I108*32)`

M164->D:$0813 **; #1 Position bias (1/[Ix08*32] cts)**

- This register contains the offset specified in the axis definition command `#1->X + <offset>`
- The online command **{axis}={constant}**or the motion program command **PSET** adds the specified offset to the existing M164 offset: `M164 = M164 + <new_offset>.`

  To read this register in counts: `P164 = M164 / (I108*32)`

M165->L:$081F **; &1 X-axis target position (engineering units)**

- M165 contains the programmed axis position through a motion program, **X10** for example, in engineering units. It also gets updated by the online command "**{axis}={constant}**" or the motion program command **PSET**.

M166->X:$0033,0,24,S **; #1 Actual velocity (1/[Ix09*32] cts/cyc)**

- M166 is the actual velocity register. For display purposes use the Motor filtered actual velocity, M174

To read this register in cts/msec: `P166 = M166 * 8388608 / (I109 * 32 * I10 * (I160+1))`

`M167->D:$002D` **; #1 Present master ((handwheel) pos (1/[Ix07*32] cts**
; of master or (1/[Ix08*32] cts of slaved motor)

- M167 is related to the master/slave relationship set through Ix05 and Ix06. It contains the present number of counts the master. To read this register in counts:

```
P167 = M167 / (I108*32)
          or
P167 = M167 / (I107*32)
```

`M169->D:$0046` **; #1 Compensation correction**

- Calculated leadscrew compensation correction according to actual position (M162) and the leadscrew compensation table set through the "**define comp"** command.

To read this register in counts: `P169 = M169 / (I108*32)`

`M172->L:$082B` **; #1 Variable jog position/distance (counts)**

- Contains the distance for the J=* command

Example:
```
M172=2000          J=*    ;Jog to position 2000 encoder counts
```

`M173->Y:$0815,0,24,S` **; #1 Encoder home capture offset (counts)**

- Contains the home offset from the reset/power-on position. Important for the capture/compare features.

Example:

```
If (M117=1)
  P103=M103-M173          ; Captured position minus offset
endif
```

`M174->Y:$082A,24` **; #1 filtered actual velocity (1/[Ix09*32] cts/servo cycle)**

- These registers contain the actual velocities averaged over the previous 80 real-time interrupt periods (80*[I8+1] servo cycles); good for display purposes.

To read this register in cts/msec: `P174 = M174 * 8388608 / (I109 * 32 * I10 * (I160+1))`

`M175->D:$0840` **; #1 following error (1/[Ix08*32] cts)**

- Following error is the difference between motor desired and measured position at any instant. When the motor is open-loop (killed or enabled), following error does not exist and PMAC reports a value of 0.

$$P176 = \frac{M161 - M162 + M164 + M169 - M175 * M167}{I108 * 32}$$

To read this register in counts: `P176 = M175 / (I108*32)`

4.7 - Homing Search Moves

If PMAC is not using an absolute feedback sensor that will keep a point of reference on the machine, the axis should be homed before running a motion program or Jog commands. If a home search procedure in not performed after power-up\reset, PMAC will consider the power-up\reset position as the zero point reference.

I-variable	Description	I-variable	Description
Ix03	Motor x Position Address	Ix26	Motor x Home Offset
Ix20	Motor x Jog/Home Acceleration Time	I902, I907,..	Encoder 0 Capture Control (PMAC 1 Only)
Ix21	Motor x Jog/Home S-Curve Time	I903, I908,..	Encoder 0 Flag Select (PMAC 1 Only)
Ix23	Motor x Homing Speed & Direction	Ix25	Motor x Flag Address

The flag channel used by Ix25 must match the position feedback channel used by Ix03 (indirectly from the conversion table)

Description	M-variable	Description	M-variable
ENC capture/compare position register	Mx03	FAULT input status	Mx23
ENC 3rd channel input status	Mx19	Desired-velocity-zero bit	Mx33
HMFL input status	Mx20	In-position bit	Mx40
-LIM input status	Mx21	Home-complete bit	Mx45
+LIM input status	Mx22	Encoder home capture offset (counts)	Mx73

Home commands could be issued on the terminal window, a Motion Program or a PLC Program:

```
-       HOME1..8        ;Home axis 1 to 8 in a Motion Program. Program is halted until home is completed.
-       #1HM            ;Online command for homing motor #1 from the terminal window.
-       CMD"#1HM"       ;Online command for homing motor #1 from a PLC program.
        while (…)       ;If a command statement is used in a PLC, the lines after must have a while
        endwhile        ;loop waiting for the home procedure to complete (see main PMAC manual for details).
```

HOMEZ is very similar to these HOME commands but no motion will result in this kind of home search. PMAC will determine the zero reference home position in the place where the axes are found when HOMEZ is issued.

4.8 - Command and Send statements

Using the COMMAND or CMD statement, online commands could be issued from a PLC or Motion program having the same result as if they were issued from a host computer or a terminal window. Certain online commands might not be valid when issued from a running program. For example, a Jog command to a motor part of a coordinate system running a motion program will be invalid. It is a good idea to have I6 not set to 2 in early development so you will know when PMAC has rejected such a command. Setting I6 to 2 in the actual application can prevent program hangup from a full response queue, or from disturbing the normal host communications protocol.
Messages to a host computer or terminal window could be issued using the SEND command.
If there is no host on the port to which the message is sent, or the host is not ready to read the message, the message is left in the queue. If several messages back up in the queue this way, the program issuing the messages will halt execution until the messages are read. This is a common mistake when the SEND command is used outside of an Edge-Triggered condition in a PLC program. On the serial port, it is possible to send messages to a non-existent host by disabling the port handshaking with I1=1.
If a program, particularly a PLC program, sends messages immediately on power-up/reset, it can confuse a host-computer program (such as the PMAC Executive Program) that is trying to "find" PMAC by querying it and looking for a particular response.
It is possible, particularly in PLC programs, to order the sending of messages or command statements faster than the port can handle them. This will almost always happen if the same **SEND** or **CMD** command is executed every scan through the PLC. For this reason, it is good practice to have at least one of the conditions that causes the **SEND** or **CMD** command to execute to be set false immediately to prevent execution of this **SEND** or **CMD** command on subsequent scans of the PLC.

5.0 - Motion Programs

PMAC can hold up to 256 motion programs at one time. Any coordinate system can run any of these programs at any time, even if another coordinate system is already executing the same program. PMAC can run as many motion programs simultaneously as there are coordinate systems defined on the card (up to 8). A motion program can call any other motion program as a subprogram, with or without arguments.
PMAC's motion program language is perhaps best described as a cross between a high-level computer language like BASIC or Pascal, and "G-Code" (RS-274) machine tool language. In fact, it can accept straight "G-Code" programs directly (provided it has been set up properly). It has the calculational and logical constructs of a computer language, and move specification constructs very much like machine tool languages. Numerical values in the program can be specified as constants or expressions.
Motion or PLCs programs are entered in any text file to be downloaded afterwards to PMAC. PEWIN provides a built-in text editor for this purpose but any other text editor could conveniently be used. Once the code has been written it can be downloaded to PMAC using PEWIN.
All PMAC commands can be issued from any terminal window communicating with PMAC. Online commands allow, for example, to jog motors, change variables, report variables values, start and stop programs, query for status information and even write short programs and PLCs. In fact, the downloading process is just a sequence of valid PMAC commands sent line by line from a particular text file.

5.1 - How PMAC Executes a Motion Program

Basically, a PMAC program exists to pass data to the trajectory generator routines that compute the series of commanded positions for the motors every servo cycle. The motion program must be working ahead of the actual commanded move to keep the trajectory generators "fed" with data.
PMAC processes program lines either zero, one or two moves (including **DWELL**s and **DELAY**s) ahead. Calculating one move ahead is necessary in order to be able to blend moves together; calculating a second move ahead is necessary if proper acceleration and velocity limiting is to be done, or a three-point spline is to be calculated (**SPLINE** mode).
For linear blended moves with I13 (move segmentation time) equal to zero (disabled), PMAC calculates two moves ahead, because the velocity and acceleration limits are enabled here. In all other cases, PMAC is calculating one move ahead.

no moves ahead	two moves ahead	one move ahead
RAPID	LINEAR with I13=0	LINEAR with I13>0
HOME	SPLINE1	CIRCLE
DWELL		PVT
"b1s" (step through the program)		
Ix92=1 (blending disabled)		

When a Run command is given, and every time the actual execution of programmed moves progresses into a new move, a flag is set saying it is time to do more calculations in the motion program for that coordinate system. At the next RTI, if this flag is set, PMAC will start working through the motion program processing each command encountered. This can include multiple modal statements, calculation statements, and logical control statements. Program calculations will continue (which means no background tasks will be executed) until one of the following conditions occurs:

1. The next move, a DWELL command or a **PSET** statement is found and calculated
2. End of, or halt to the program (e.g. **STOP**) is encountered
3. Two jumps backward in the program (from **ENDWHILE** or **GOTO**) are performed
4. A **WAIT** statement is encountered (usually in a **WHILE** loop)

If calculations stop on condition 1 or 2, the calculation flag is cleared, and will not be set again until actual motion progresses into the next move (1) or a new Run command is given (2). If calculations stop on conditions 3 or 4, the flag remains set, so calculations will resume at the next RTI. In these cases you have an "empty" (no-motion) loop, the motion program acts much like a PLC 0 during this period
If PMAC ever cannot finish calculating the trajectory for a move by the time execution of that move is supposed to begin, PMAC will abort the program, showing a run-time error in its status word.

5.2 - Coordinate Systems

A coordinate system in PMAC is a grouping of one or more motors for the purpose of synchronizing movements. A coordinate system (even with only one motor) can run a motion program; a motor cannot. PMAC can have up to 8 coordinate systems, addressed as &1 to &8, in a very flexible fashion (e.g. 8 coordinate systems of 1 motor each, 1 coordinate system of 8 motors, 4 coordinate systems of two motors each, etc.).
In general, if you want certain motors to move in a coordinated fashion, put them in the same coordinate system. If you want them to move independently of each other, put them in separate coordinate systems. Different coordinate systems can run separate programs at different times (including overlapping times), or even run the same program at different (or overlapping) times.
A coordinate system must first be established by assigning axes to motors in 'Axis Definition Statements'. A coordinate system must have at least one motor assigned to an axis within that system, or it cannot run a motion program, even non-motion parts of it. When a program is written for a coordinate system, if simultaneous motions are desired of multiple motors, their move commands are simply put on the same line, and the moves will be coordinated.

5.2.1 - Axis definitions

An axis is an element of a coordinate system. It is similar to a motor, but not the same thing. An axis is referred to by letter. There can be up to 8 axes in a coordinate system, selected from X, Y, Z, A, B, C, U, V, and W. An axis is defined by assigning it to a motor with a scaling factor and an offset (X, Y, and Z may be defined as linear combinations of three motors, as may U, V, and W). The variables associated with an axis are scaled floating-point values.
In the vast majority of cases, there will be a one-to-one correspondence between motors and axes. That is, a single motor is assigned to a single axis in a coordinate system. Even when this is the case, however, the matching motor and axis are not completely synonymous. The axis is scaled into engineering units, and deals only with *commanded* positions. Except for the **PMATCH** function, calculations go only from axis commanded positions to motor commanded positions, not the other way around.
More than one motor may be assigned to the same axis in a coordinate system. This is common in gantry systems, where motors on opposite ends of the cross-piece are always trying to do the same movement. By assigning multiple motors to the same axis, a single programmed axis move in a program causes identical commanded moves in multiple motors. This is commonly done with two motors, but up to eight motors have been used in this manner with PMAC. Remember that the motors still have independent servo loops, and that the *actual* motor positions will not necessarily be exactly the same.
An axis in a coordinate system can have no motors attached to it (a "phantom" axis), in which case programmed moves for that axis cause no movement, although the fact that a move was programmed for that axis can affect the moves of other axes and motors. For instance, if sinusoidal profiles are desired on a single axis, the easiest way to do this is to have a second, "phantom" axis and program circularly interpolated moves.

5.2.2 - Axis Definition Statements

A coordinate system is established by using axis definition statements. An axis is defined by matching a motor (which is numbered) to one or more axes (which are specified by letter).
The simplest axis definition statement is something like **#1->X**. This simply assigns motor #1 to the X axis of the currently addressed coordinate system. When an X axis move is executed in this coordinate system, motor #1 will make the move. The axis definition statement also defines the scaling of the axis' user units. For instance, **#1->10000X** also matches motor #1 to the X axis, but this statement sets 10,000 encoder counts to one X-axis user unit (e.g. inches or centimeters). This scaling feature is almost universally used. Once the scaling has been defined in this statement, the user can program the axis in engineering units without ever needing to deal with the scaling again.

Permitted Axis Names: **X,Y,Z,U,V,W,A,B,C**

X,Y,Z: Traditionally Main Linear Axes
- Matrix Axis Definition
- Matrix Axis Transformation
- Circular Interpolation
- Cutter Radius Compensation

A,B,C: Traditionally Rotary Axes
(A rotates about X, B about Y, C about Z)
- Position Rollover (Ix27)

U,V,W: Traditionally Secondary Linear Axes
- Matrix Axis Definition

5.3 - Writing a MOTION PROGRAM

1) Open a program buffer with **OPEN PROG {constant}** where **{constant}** is an integer from 1 to 32767 representing the motion program to be opened.
2) Motion Programs 1000, 1001, 1002 and 1003 could contain G-codes, M-codes, T-codes and D-codes for machine tool "G-codes" or RS-274 programming method. Still these buffers could be used for general PMAC code programming as long as G-codes programming is not needed in PMAC.
3) PMAC can hold up to 256 motion programs at one time. For continuous execution of programs larger than PMAC's memory space a special PROG0, the rotary motion program buffers, allow for the downloading of program lines during the execution of the program and for the overwriting of already executed program lines.
4) The **CLEAR** command empties the currently opened program, PLC, rotary, etc. buffer.
5) Many of the statements in PMAC motion programs are modal in nature. These include move modes, which specify what type of trajectory a move command will generate; this category includes **LINEAR**, **RAPID**, **CIRCLE**, **PVT**, and **SPLINE**.
6) Moves can be specified either incrementally (distance) or absolutely (location) -- individually selectable by axis -- with the **INC** and **ABS** commands. Move times (**TA**, **TS**, and **TM**) and/or speeds (**F**), are implemented in modal commands. Modal commands can precede the move commands they are to affect, or they can be on the same line as the first of these move commands.
7) The move commands themselves consist of a one-letter axis-specifier followed by one or two values (constant or expression). All axes specified on the same line will move simultaneously in a coordinated fashion on execution of the line; consecutive lines execute sequentially (with or without stops in between, as determined by the mode). Depending on the modes in effect, the specified values can mean, destination, distance, and/or velocity.
8) If the move times (**TA**, **TS**, and **TM**) and/or speeds (**F**) are not specifically declared in the motion program the default parameters from the I-variables Ix87, Ix88 and Ix89 will be used instead. **You are strongly encouraged *not* to rely on these parameters and to declare your move times in the program. This will keep your move parameters with your move commands, lessening the chances of future errors, and making debugging easier.**
9) In a motion program, PMAC has **WHILE** loops and **IF..ELSE** branches that control program flow. These constructs can be nested indefinitely. In addition, there are **GOTO** statements, with either constant or variable arguments (the variable **GOTO** can perform the same function as a Case statement). **GOSUB** statements (constant or variable destination) allow subroutines to be executed within a program. **CALL** statements permit other programs to be entered as subprograms. Entry to the subprogram does not have to be at the beginning -- the statement **CALL 20.15000** causes entry into Program 20 at line **N15000**. **GOSUBs** and **CALLs** can be nested only 15 deep.
10) The **CLOSE** statement closes the currently **OPEN**ed buffer. This should be used immediate after the entry of a motion, PLC, rotary, etc. buffer. If the buffer is left open, subsequent statements that are intended as on-line commands (e.g. **P1=0**) will get entered into the buffer instead. It is good practice to have **CLOSE** at the beginning and end of any file to be downloaded to PMAC. When PMAC receives a **CLOSE** command, it automatically appends a **RETURN** statement to the end of the open program buffer. If *any* PROGRAM or PLC in PMAC is improperly structured (e.g. no **ENDIF** or **ENDWHILE** to match an **IF** or **WHILE**), PMAC will report an ERR003 at the **CLOSE** command for any buffer until the problem is fixed.

Example:

```
close                   ; Close any buffer opened
delete gather           ; Erase unwanted gathered data
undefine all            ; Erase coordinate definitions in all coordinate systems

#1->2000X               ; Motor #1 is defined as axes X

OPEN PROG 1 CLEAR       ; Open buffer to be written
LINEAR                  ; Linear interpolation
INC                     ; Incremental mode
TA100                   ; Acceleration time is 100 msec
TS0                     ; No S-curve acceleration component
F50                     ; Feedrate is 50 Units per Ix90 msec
X1                      ; One unit of distance, 2000 encoder counts
CLOSE                   ; Close written buffer, program one
```

5.4 - Running a MOTION PROGRAM

1) Select the Coordinate System where the motion program will be running under. This is done by issuing the command & followed by the coordinate system number, like **&1** for the coordinate system one.
2) Select the program that you want to run with the **B{constant}** command, where the **{constant}** represents the number of the motion program buffer. You *must* use the **B** command to change motion programs, and after *any* motion program buffer has been opened. You do not have to use it if you are repeatedly running the same motion program without modification; when PMAC finishes executing a motion program, the program counter for the coordinate system is automatically set to point to the beginning of that program, ready to run it again.
3) Once you are pointing to the motion program you wish to run, you may issue the command to start execution of the program. If you wish continuous execution of the program, use the **R** command (**<CTRL-R>** for all coordinate systems simultaneously). The program will execute all the way through unless stopped by command or error condition.
4) If you wish to execute just one move, or a small section of the program, use the **S** command (**<CTRL-S>** for all coordinate systems simultaneously). The program will execute to the first move **DWELL**, or **DELAY**, or if it first encounters a **BLOCKSTART** command, it will execute to the **BLOCKSTOP** command.
5) When a run or step command is issued, PMAC checks the coordinate system to make sure it is in proper working order. If it finds anything in the coordinate system is not set up properly, it will reject the command, sending a *<BELL>* command back to the host. If I6 is set to 1 or 3, it will report an error number as well telling the reason the command was rejected. PMAC will reject a run or step command for any of the following reasons:

   - A motor in the coordinate system has both overtravel limits tripped (ERR010)
   - A motor in the coordinate system is currently executing a move (ERR011)
   - A motor in the coordinate system is not in closed-loop control (ERR012)
   - A motor in the coordinate system in not activated {Ix00=0} (ERR013)
   - There are no motors assigned to the coordinate system (ERR014)
   - A fixed (non-rotary) motion program buffer is open (ERR015)
   - No motion program has been pointed to (ERR016)
   - After a / or \ stop command, a motor in the coordinate system is not at the stop point (ERR017)

6) Before starting the program is convenient to issue a CTRL+A command to PMAC to ensure that all the motors will be potentially in closed loop and that all previous motions are aborted. Also, if in doubt, the functioning of each motor could be check individually prior to run a program by means of Jog commands. For example, "`#1J^2000`" will make motor #1 move 2000 encoder counts and that would be a way to confirm if the motors are able to run motion programs or not.
7) All motors in the addressed coordinate system have to be ready to run a motion program. Depending on Ix25, even if one motor defined in the coordinate system is not closing the loop, all motors in the coordinate system could be brought down impeding of running any motion program.
8) Sometimes the feedrate override for the current addressed coordinate system is set at zero and no motion will occur in result of this. Check the feedrate override parameter by issuing a "`&1%`" command on the terminal window (replace 1 for the appropriate coordinate system number). If is zero or too low, set it to an appropriate value. The "`&1%100`" command will set it to 100 %.
9) For troubleshooting purposes it is a good technique to change the feedrate override to a lower than 100% value. If the program is run for the first time, a preceding "`%10`" command could be issued to run the motion program in "slow" motion. Running the program slowly will allow observing the programmed path more clearly, it will demand less calculation time from PMAC and it will prevent damages due to potentially wrong acceleration and/or feedrate parameters.
10) A motion program could be stopped by sending a "`&1a`" or, for simplicity, a CTRL+A command which will stop any motion taking place in PMAC.
11) If the motion of any axis becomes uncontrollable and is desired to be stopped, a CTRL+K command could be issued "killing" all the motors in PMAC (disabling the amplifier enable line if connected). However, the motor will come to a stop in an uncontrollable way and might proceed to move due to its own inertia.
12) A motion program could also be stop by issuing a CTRL+Q command. The last programmed moves in the buffer will be completed before the program quits execution. It could be resumed by issuing an "R" command alone, without first pointing to the beginning of the buffer by the "B" command.

5.5 - Subroutines and Subprograms

It is possible to create subroutines and subprograms in PMAC motion programs to create well-structured modular programs with re-usable subroutines. The **GOSUBx** command in a motion program causes a jump to line label **Nx** of the same motion program. Program execution will jump back to the command immediately following the **GOSUB** when a **RETURN** command is encountered. This creates a subroutine.
The **CALLx** command in a motion program causes a jump to PROG x, with a jump back to the command immediately following the **CALL** when a **RETURN** command is encountered. If **x** is an integer, the jump is to the beginning of PROG x; if there is a fractional component to **x**, the jump is to line label **N(y*100,000)**, where **y** is the fractional part of **x**. This structure permits the creation of special subprograms, either as a single subroutine, or as a collection of subroutines, that can be called from other motion programs.
The **PRELUDE** command allows creating an automatic subprogram call before each move command or other letter-number command in a motion program.

5.5.1 - Passing Arguments to Subroutines

These subprogram calls are made more powerful by use of the **READ** statement. The **READ** statement in the subprogram can go back up to the calling line and pick off values (associated with other letters) to be used as arguments in the subprogram. The value after an A would be placed in variable Q101 for the coordinate system executing the program, the value after a B would be placed in Q102, and so on (Z value goes in Q126). Letters "N" or "O" cannot be passed.
This structure is particularly useful for creating machine-tool style programs, in which the syntax must consist solely of "letter-number" combinations in the parts program. Since PMAC treats the G, M, T, and D codes as special subroutine calls, the READ statement can be used to let the subroutine access values on the part-program line after the code.
The **READ** statement also provides the capability of seeing what arguments have actually been passed. The bits of Q100 for the coordinate system are used to note whether arguments have been passed successfully; bit 0 is 1 if an A argument has been passed, bit 1 is 1 if a B argument has been passed, and so on, with bit 25 set to 1 if a Z argument has been passed. The corresponding bit for any argument not passed in the latest subroutine or subprogram call is set to 0.
Example:

```
close delete gather undefine all
#1->2000X
open prog1 clear
LINEAR INC TA100 TS0 F50        ;Mode and timing parameters
gosub 100 H10                   ;Subroutine call passing parameter H with value 10
return                          ;End of the main program section (execution ends)
n100                            ;Subroutines section. First subroutine labeled 100
read(h)                         ;Read the "H" parameter value passed
IF (Q100 & $80 > 0)             ;If the "H" parameter has been passed …
      X(Q108)                   ;Use the "H" parameter value contained in Q108
endif
return                          ;End of the subroutine labeled 100
close                           ;End of the motion program code
```

5.5.2 - G, M, T, and D-Codes (Machine-Tool Style Programs)

PMAC permits the execution of machine-tool style RS-274 ("G-Code") programs by treating G, M, T, and D codes as subroutine calls. This permits the machine tool manufacturer to customize the codes for his own machine, but it requires the manufacturer to do the actual implementation of the subroutines that will execute the desired actions. When PMAC encounters the letter G with a value in a motion program, it treats the command as a CALL to motion program 10n0, where n is the hundreds' digit of the value. The value without the hundred's digit (modulo 100 in mathematical terms) controls the line label within program 10n0 to which operation will jump -- this value is multiplied by 1000 to specify the number of the line label. When a return statement is encountered, it will jump back to the calling program.
For example, **G17** will cause a jump to **N17000** of PROG 1000; **G117** will cause a jump to **N17000** of PROG 1010; **G973.1** will cause a jump to **N73100** of PROG 1090.
M-codes are the same, except they use PROG 10n1; T-codes use PROG 10n2; D-codes use PROG 10n3.
Most of the time, these codes have numbers within the range 0 to 99, so only PROGs 1000, 1001, 1002, and 1003 are required to execute them. For those who want to extend code numbers past 100, PROGs 1010, 1011, etc. will be required to execute them.

5.6 - Linear blended moves

- The move time is set directly by TM or indirectly based on the the distances and feedrate (F) parameters set:

```
TM100
X3 Y4
```

or

```
FRAX(X,Y)
X3 Y4 F50
```

$$;TM = \frac{I190 \cdot \sqrt{3^2 + 4^2}}{50} = \frac{5000}{50} = 100 \text{ msec}$$

- If the move time above calculated is less than the TA time set, the move time used will be the TA time instead. In this case, the programmed TA (or `2*TS` if `TA<2*TS`) results in the minimum move time of a linearly interpolated move.

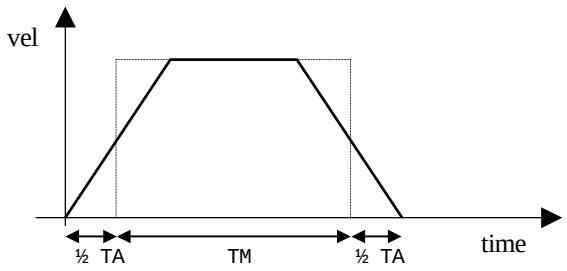
- If the TA programmed results to be less than twice the TS programmed, `TA<2*TS`, the TA time used will be `2*TS` instead.

- The acceleration time TA of a blended move cannot be longer than two times the previous TM minus the previous TA, otherwise the value 2*(TM-½ TA) will be used as the current TA instead.

- The safety variables Ix16 and Ix17 will override these parameters if they are found to violate the programmed limits.

- If TM < TA, TM = TA
- If TA < 2*TS, TA = 2*TS
- If TA_{i+1} > 2*(TM_i - ½ TA_i), TA_{i+1} = 2*(TM_i - ½ TA_i)

Example:

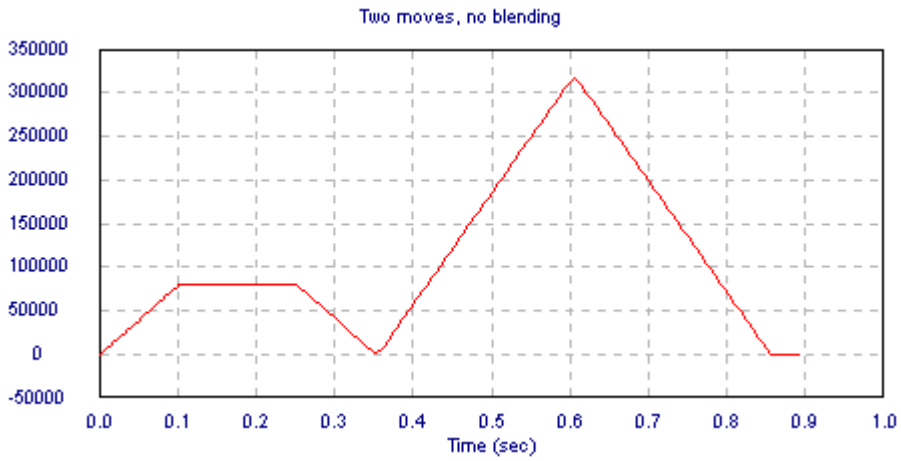


To illustrate how PMAC blends linear moves, a series of velocity Vs time profiles will be shown. The moves are defined with zero S-curve components. The concepts described here could be used for non-zero S-curve linear moves.

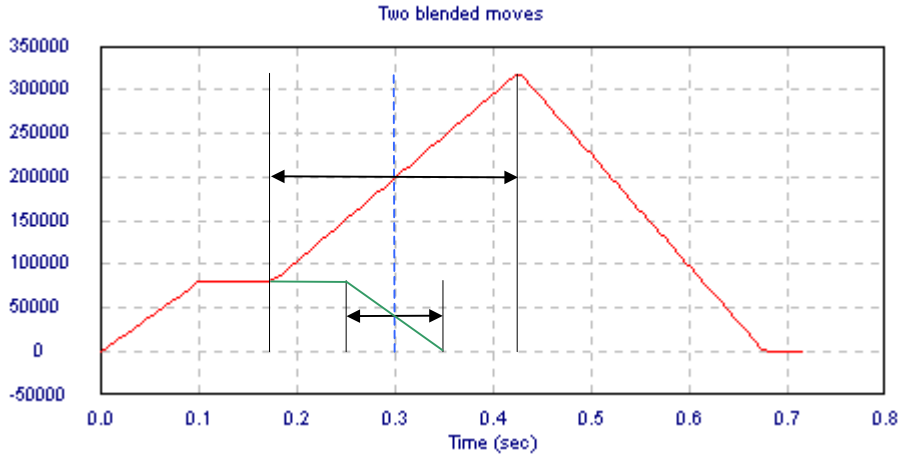
1) Lets consider the following motion program code:

```
close
delete gather
undefine all
&1
#1->2000x

OPEN PROG 1 CLEAR
        LINEAR          ; Linear mode
        INC             ; Incremental mode
        TA100           ; The acceleration time is 100 msec, TA1
        TS0             ; No S-curve component
        TM250           ; Move time is 250 msec, TM1
        X10             ; Move distance is 10 units, 20000 counts
        TA250           ; Acceleration \ deceleration of the blended move is 250 msec , TA2
        X40             ; Move distance is 40 units, 80000 counts
CLOSE
```

2) The two move commands are plot without blending, placing a DWELL0 command in between the two moves:



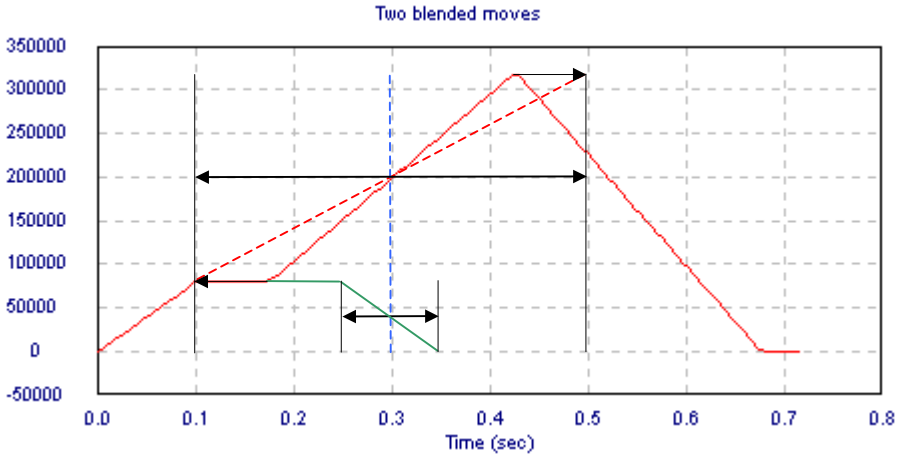
3) The two moves are now plot with the blending mode activated. To find out the blending point we trace straight lines through the middle point of each acceleration lines of both velocity profiles:



5.6.1 - Observations:

1) The total move time is given by: $\frac{TA_1}{2} + TM_1 + TM_2 + \frac{TA_2}{2} = 50 + 250 + 250 + 125 = 675\,\text{msec}$

2) The acceleration of the second blended move could only be extended up to a certain limit, 2*(TM- ½ TA):

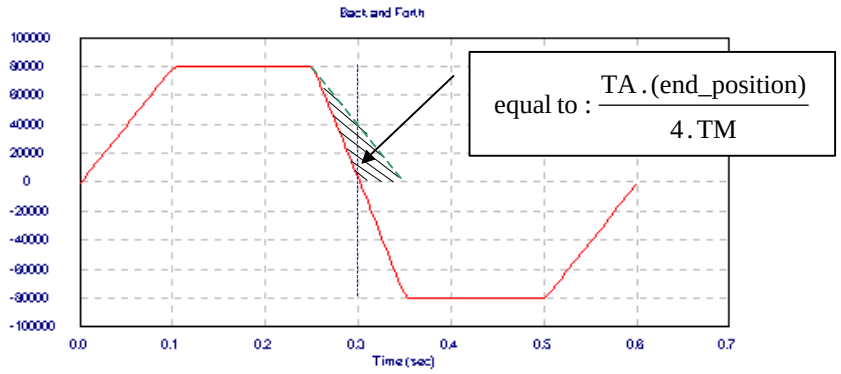


PMAC looks two moves ahead of actual move execution to perform its acceleration limit, and can recalculate these two moves to keep the accelerations under the Ix17 limit. However, there are cases where more than two moves, some much more than two, would have to be recalculated in order to keep the accelerations under the limit. In these cases, PMAC will limit the accelerations as much as it can, but because the earlier moves have already been executed, they cannot be undone, and therefore the acceleration limit will be exceeded.

3) When performing a blended move that involve a change of direction, the end point might no be reached.

Example:

```
TA100
TM250

X10             ; This would reach only to position = 10 - 100.10/4.250 = 9

X-10
```



In order to reach the desired position, since the move involves a change in direction and stop, simply place a DWELL0 command between moves. This command will disable blending for that particular move:

```
TA100
TM250
X10
DWELL0
X-10
```

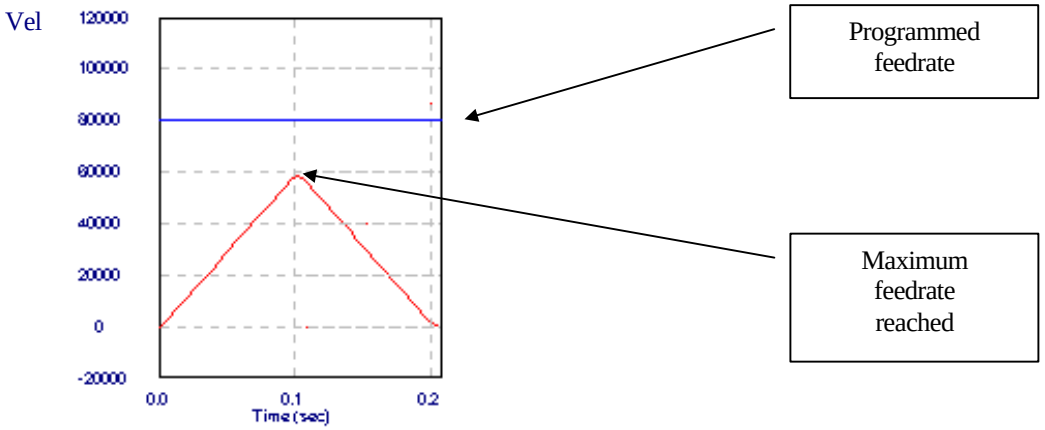
4) Since the value of TA determines the minimum time in which a programmed move could be executed, it could limit the maximum move velocity and therefore the programmed feedrate might not be reached. This is seen in "triangular" velocity profile moves types, especially when a sequence of short distance moves is programmed.

Example:

```
close
delete gather
undefine all
&1
#1->2000X
I190=1000

OPEN PROG 1 CLEAR
        LINEAR          ; Linear mode
        INC             ; Incremental mode
        TA100           ; Acceleration time is 100 msec, TA1
        TS0             ; No S-curve component
        F40             ; Feedrate is 40 length_units / second
        X3              ; TM = 3.I190/40 = 3000/40 = 75 msec
CLOSE
```

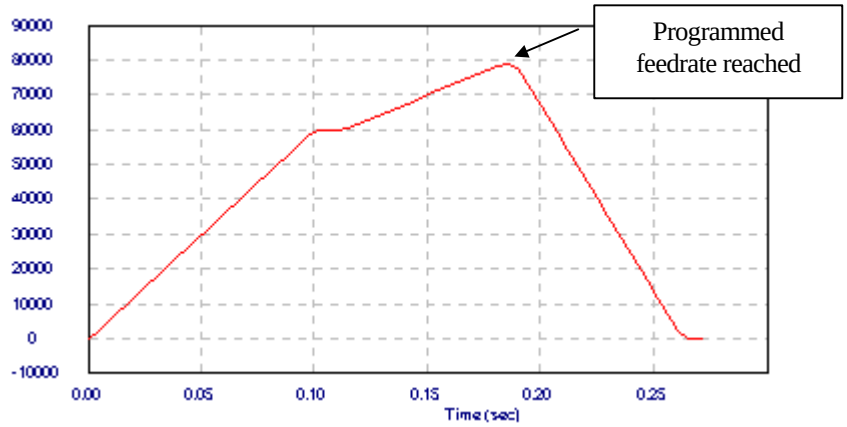
Since the calculated TM for the given feedrate is 75 msec and the programmed TA for this move is 100 msec, the TM used will be100 msec instead. This yields the following feedrate value instead of the programmed one:

$$F = \frac{3.I190}{100} = \frac{3000}{100} = 30 \frac{\text{units of distance}}{\text{second}}$$

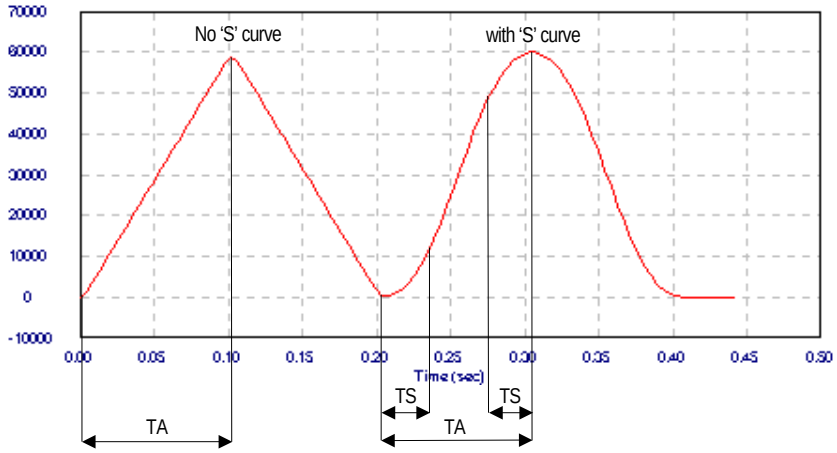


To be able to reach the desired velocity, a longer move could be performed split into two sections. The first move will be executed using a suitable TA to get the motor to move from rest. The second move will have a lower acceleration time TA in order to decrease the move time TM and so reach the programmed feedrate.

```
OPEN PROG 1 CLEAR
        LINEAR
        INC
        TS0
        F40
        TA100
        X3
        TA75
        X3
CLOSE
```

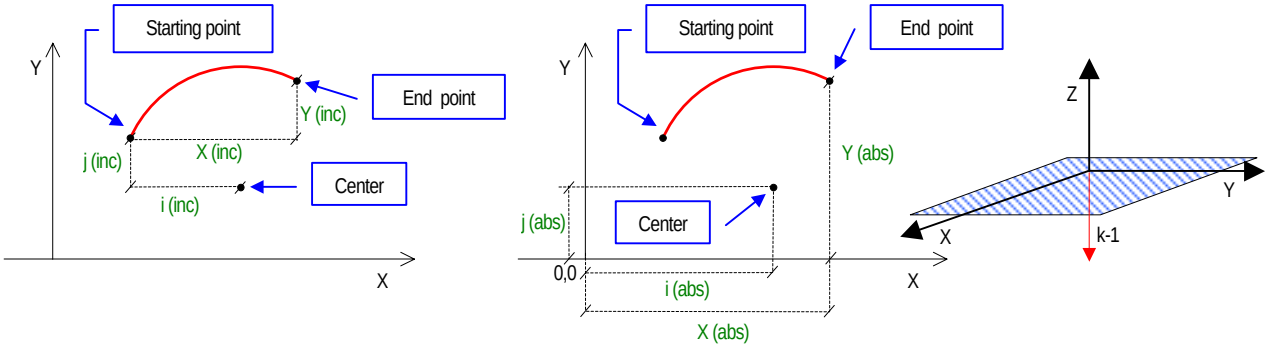


5) All the previous analysis was performed assuming a zero ‘S” curve component. A move executed with an ‘S” curve component will be similar in shape but with rounded sections at the beginning and end of the acceleration lines.



5.7 - Circular Interpolation

PMAC allows circular interpolation on the X, Y, and Z axes in a coordinate system. As with linear blended moves, **TA** and **TS** control the acceleration to and from a stop, and between moves. Circular blended moves can be either feedrate-specified (**F**) or time-specified (**TM**), just as with linear moves. It is possible to change back and forth between linear and circular moves without stopping. This is accomplish by entering the command **LINEAR** when linear interpolation is needed and **CIRCLE1** or **CIRCLE2** for circular interpolation.



1) PMAC performs arc moves by segmenting the arc and performing the best cubic fit on each segment. I-variable I13 determines the time for each segment. I13 *must* be set greater than zero to put PMAC into this segmentation mode in order for arc moves to be done. If I13 is set to zero, circular arc moves will be done in linear fashion.
The practical range of I13 for the circular interpolation mode is 5-10 msec. A value of 10 msec is recommended for most applications, a lower than 10 msec I13 value will improve the accuracy of the interpolation (calculating points of the curve more often) but will also consume more of the PMAC's total computational power.
2) When PMAC is automatically segmenting moves (I13 > 0), which is required for Circular Interpolation, the Ix17 accelerations limits and the Ix16 velocity limits are not observed.
3) Any axes used in the circular interpolation are automatically feedrate axes for circular moves, even if they were not so specified in an **FRAX** command. Other axes may or may not be feedrate axes. Any non-feedrate axes commanded to move in the same move command will be linearly interpolated so as to finish in the same time. This permits easy helical interpolation.
4) The plane for the circular arc must have been defined by the **NORMAL** command (the default -- **NORMAL K-1** -- defines the XY plane). This command can only define planes in XYZ-space, which means that only the X, Y, and Z axes can be used for circular interpolation. Other axes specified in the same move command will be interpolated linearly to finish in the same time. The most commonly used planes are:

```
NORMAL K-1      ; XY plane -- equivalent to G17
NORMAL J-1      ; ZX plane -- equivalent to G18
NORMAL I-1      ; YZ plane -- equivalent to G19
```

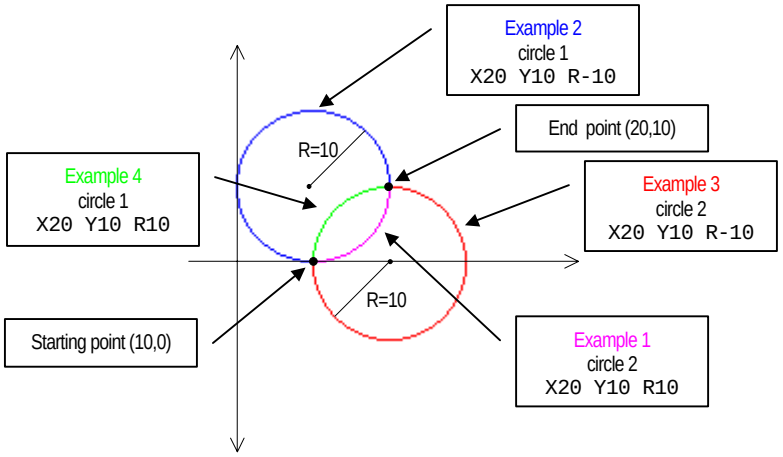
5) To put the program in circular mode, use the program command **CIRCLE1** for clockwise arcs (G02 equivalent) or **CIRCLE2** for counterclockwise arcs (G03 equivalent). **LINEAR** will restore you to linear blended moves. Once in circular mode, a circular move is specified with a move command specifying the move endpoint and either the vector to the arc center or the distance (radius) to the center. The endpoint may be specified either as a position or as a distance from the starting point, depending on whether the axes are in absolute (**ABS**) or incremental (**INC**) mode (individually specifiable).

```
X{Data} Y{Data} R{Data}          ;Radius of the circle is given
X{Data} Y{Data} I{Data} J{Data}  ;Center coordinates of the circle are given
```

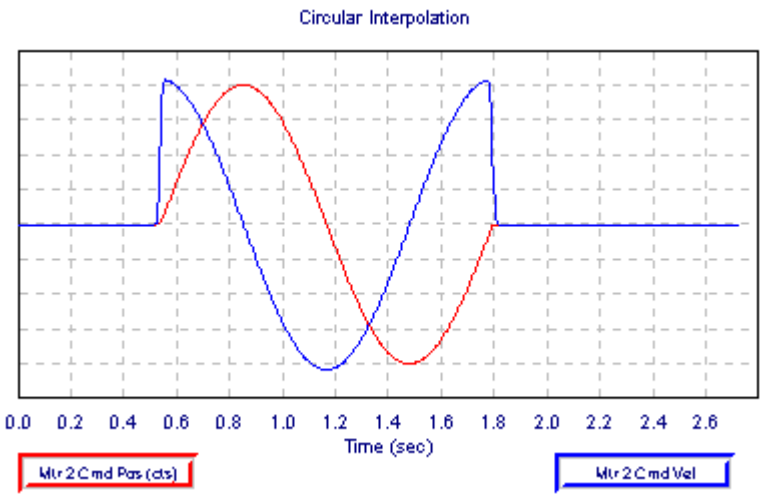
6) If the vector method of locating the arc center is used, the vector is specified by its I, J, and K components ('I' specifies the component parallel to the X axis, 'J' to the Y axis, and 'K' to the 'Z' axis). This vector can be specified as a distance from the starting point (i.e. incrementally), or from the XYZ origin (i.e. absolutely). The choice is made by specifying 'R' in an ABS or INC statement (e.g. **ABS (R)** or **INC (R)**). This affects I, J, and K specifiers together. (**ABS** and **INC** without arguments affect all axes, but leave the vectors unchanged). The default is for incremental vector specification.

7) PMAC's convention is to take the short arc path if the **R** value is positive, and the long arc path if **R** is negative:

- If the value of R is positive, the arc to the move endpoint is the short route (<=180 degrees)
- If the value of R is negative, the arc to the move endpoint is the long route (>=180 degrees)



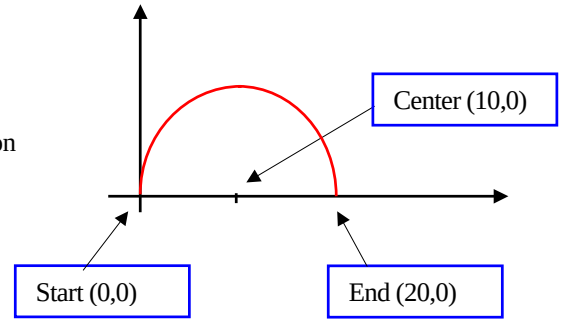
8) When performing a circular interpolation, the individual axes describe a position Vs time profile close to a sine and cosine shape. This is also true for their velocity and acceleration profiles. Therefore, circular interpolation makes an ideal feature to described trigonometric profiles. Further, the period (and so frequency) of the sine or cosine waves could be set by the total move time given by TA+TM.




```
close
delete gather
undefine all
&1
#2->2000Y    ;X is phantom
open prog1 clear
inc
inc (r)
ta300
ts0
tm1000       ;TA+TM is period
i13=10
normal k-1   ;X-Y plane
circle1      ;clockwise
x0 y0 i10    ;complete circle
close
&1b1r
```

Example:

```
I13=10              ;Move Segmentation Time
NORMAL K-1          ;XY plane
INC                 ;Incremental End Point definition
INC (R)             ;Incremental Center Vector definition
CIRCLE 1            ;Clockwise circle
X20 Y0 I10 J0       ;Arc move
```

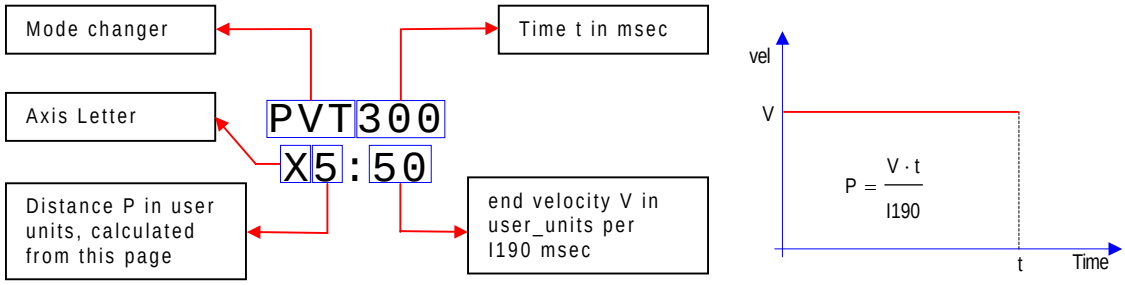


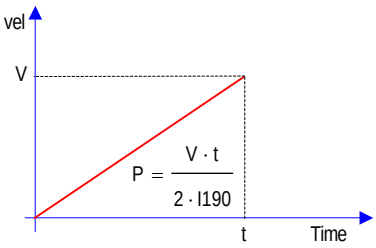
5.8 - Splined Moves

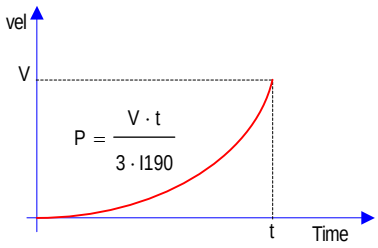
PMAC can perform cubic splines (cubic in terms of the position vs time equations) to blend together a series of points on an axis. Splining is particularly suited to "odd" (non-cartesian) geometries, such as radial tables and rotary-axis robots, where there are odd axis profile shapes even for regular "tip" movements.
In **SPLINE1** mode, a long move is split into equal-time segments, each of TA time. Each axis is given a destination position in the motion program for each segment with a normal move command line like **X1000Y2000**. Looking at the move command before this and the move command after this, PMAC creates a cubic position-vs-time curve for each axis so that there is no sudden change of either velocity or acceleration at the segment boundaries. The commanded position at the segment boundary may be "relaxed" slightly to meet the velocity and acceleration constraints.
PMAC can only work with integer (millisecond) values for the TA segment times. If a non-integer value is specified for the TA time, PMAC will automatically round it to the nearest integer. It will not report an error. This rounding will change the speeds and times for the trajectory.
At the beginning and end of a series of splined moves, PMAC automatically adds a zero-distance segment of TA time for each axis, and performs the spline between this segment and the adjacent one. This results in an S-curve acceleration to and from a stop.
PMAC's **SPLINE2** mode is very similar to the **SPLINE1** mode, except that the requirement that the TA spline segment time remain constant is removed.

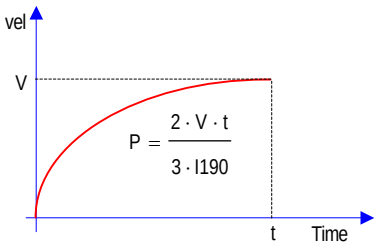
5.9 - PVT-Mode Moves

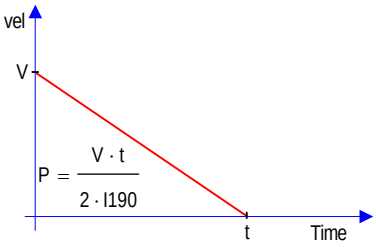
For the user who desires more direct control over the trajectory profile, PMAC offers Position-Velocity-Time (PVT) mode moves. In these moves, the user specifies the axis states directly at the transitions between moves (unlike in blended moves). This requires more calculation by the host, but allows tighter control of the profile shape. For each piece of a move, the user specifies the end position or distance, the end velocity, and the piece time.
PMAC is put in this mode with the program statement **PVT{data}**, where **{data}** is a constant, variable, or expression, representing the piece time in milliseconds. This value should be an integer; if it is not, PMAC will round it to the nearest integer. The piece time may be changed between pieces, either with another **PVT{data}** statement, or with a **TA{data}** statement. The program is taken out of this mode with another move mode statement (e.g. **LINEAR**, **RAPID**, **CIRCLE**, **SPLINE**).
A PVT mode move is specified for each axis to be moved with a statement of the form **{axis}{data}:{data}**, where **{axis}** is a letter specifying the axis, the first **{data}** is a value specifying the end position or the piece distance, depending on whether the axis is in absolute or incremental mode, respectively, and the second **{data}** is a value representing the ending velocity.
The units for position or distance are the user length or angle units for the axis, as set in the Axis Definition statement. The units for velocity are defined as length units divided by time units, where the length units are the same as those for position or distance, and the time units are defined by variable Ix90 for the coordinate system (feedrate time units). The velocity specified for an axis is a signed quantity.
From the specified parameters for the move piece, and the beginning position and velocity (from the end of the previous piece), PMAC computes the only third-order position trajectory path to meet the constraints. This results in linearly changing acceleration, a parabolic velocity profile, and a cubic position profile for the piece.
Since the user can specify (directly or indirectly) a non-zero end velocity for the move, it is not a good idea to step through a program of transition-point moves, and great care must be exercised in downloading these moves in real time. With the use of the **BLOCKSTART** and **BLOCKSTOP** statements surrounding a series of PVT moves, the last of which has a zero end velocity, it is possible to use a Step command to execute only part of a program.
The PVT mode is the most useful for creating arbitrary trajectory profiles. It provides a "building block" approach to putting together parabolic velocity segments to create whatever overall profile is desired. The diagram *PVT Segment Shapes*, below, shows common velocity segment profiles. PVT mode can create any profile that any other move mode can.
PVT mode provides excellent contouring capability, because it takes the interpolated commanded path exactly through the programmed points. It creates a path known as a "Hermite Spline". **LINEAR** and **SPLINE** modes are 2nd and 3rd-order B-splines, respectively, which pass to the inside of programmed points. Compared to PMAC's SPLINE mode, PVT produces a more accurate profile.

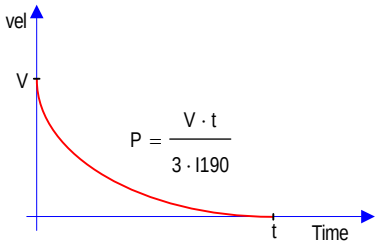


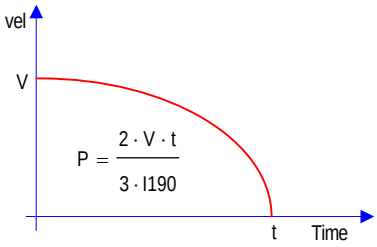


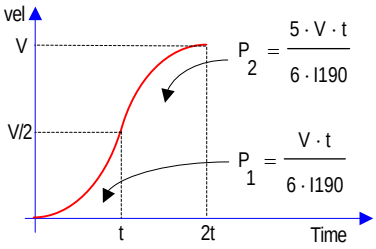


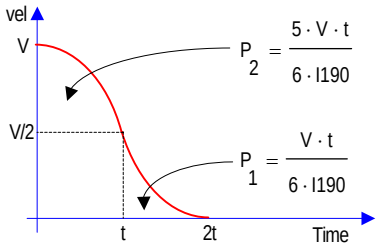


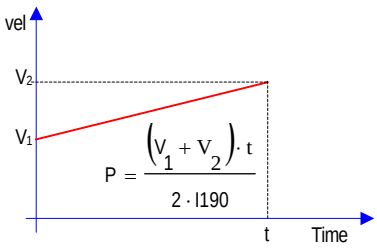








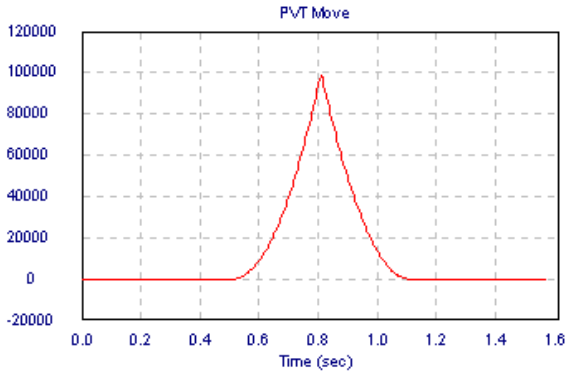




Replace I190 for the appropriate Ix90 variable according to coordinate system x.

Example:

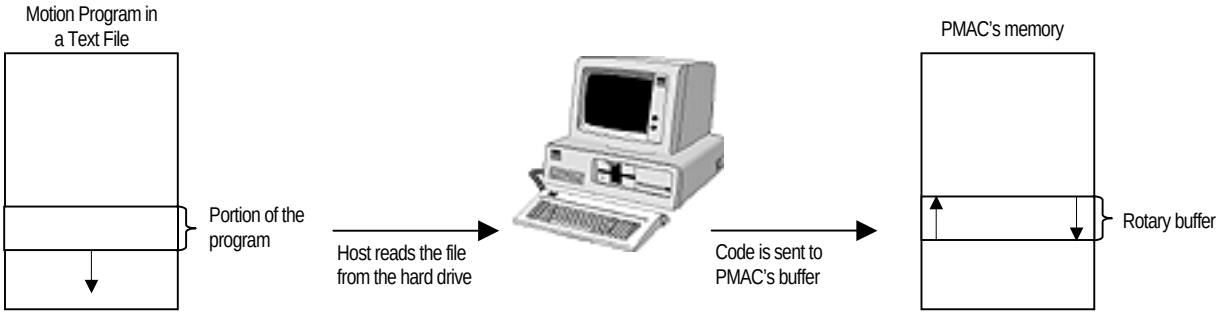
```
close delete gather undefine all
&1 #1->2000X

OPEN PROG 1 CLEAR
INC
PVT300      ;Time is 300 msec per section

X5:50       ; P = 50 user_units / I190 msec · 300 msec / 3 = 15000 / 3000 = 5 user_units

X5:0        ; P = 50 user_units / I190 msec · 300 msec / 3 = 15000 / 3000 = 5 user_units

CLOSE
```



5.10 - Other programming features

5.10.1 - Rotary Motion Program Buffers:

PMAC has a limited memory space shared for motion programs, plcs, compensation tables and gathering buffers. The rotary motion program buffers allows running motion programs larger than the available space in PMAC's memory.


Motion Program in a Text File
Portion of the program
Host reads the file from the hard drive
PMAC's memory
Code is sent to PMAC's buffer
Rotary buffer

Communication routines provided by Delta Tau have the necessary code to implement this feature in a host computer.

5.10.2 - Internal Timebase, the feedrate override

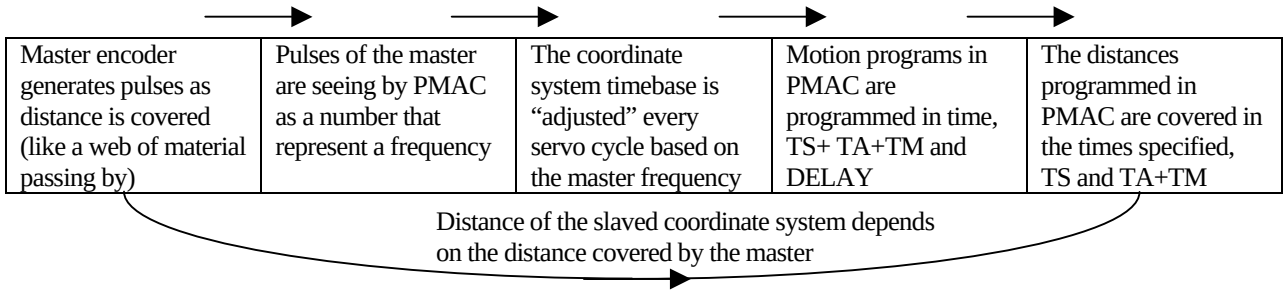
Each coordinate system has its own "time base" that helps control the speed of interpolated moves in that coordinate system. If Ix93 is set at default, this parameter could be changed by different means:

- %*n*, where $0 < n < 100$: Online or CMD command that runs all motion commands in "slow-motion".
- %*n*, where $100 < n \leq 225$: Online or CMD command that runs all motion commands proportionally faster.
- %0 : Online or CMD command that "freezes" all motions and timing in that C.S.
- %100 : Online or CMD command that restores the real-time reference (1 msec = 1 msec).
- M197 = I10 : Suggested M-variable for timebase change. Equal to I10 is 100%, equal to 0 is 0%.

The variable Ix94 controls the rate at which the timebase changes: $Ix94 = \frac{I10^2}{t \cdot 2^{23}}$, where t is the slew rate time in msec.

5.10.3 - External Time-Base Control (Electronic Cams)

The time reference of each coordinate system could be changed from the default internal reference, controlled by the "%" command and variables Mx97, to an external source (usually a frequency reference from a master encoder). A simple change of the variable Ix93 allows switching between the internal timebase and an external source. In this fashion, motion programs can be developed and tested running in real-time (internal timebase) and synchronized later to a master frequency when proven to be functional and completed.


Master encoder generates pulses as distance is covered (like a web of material passing by)
Pulses of the master are seeing by PMAC as a number that represent a frequency
The coordinate system timebase is "adjusted" every servo cycle based on the master frequency
Motion programs in PMAC are programmed in time, TS+ TA+TM and DELAY
The distances programmed in PMAC are covered in the times specified, TS and TA+TM
Distance of the slaved coordinate system depends on the distance covered by the master

The only setup part of this feature is an entry in the conversion table that will also indicate a scale factor for the maximum frequency that the master can possibly input to PMAC. This maximum frequency will represent 100% or real-time.

5.10.4 - Position Following (Electronic Gearing)

PMAC has several methods of coordinating the axes under its control to axes not under its control. The simplest method is basic position following. This is a motor-by-motor function, not a coordinate system function as time-base following is. An encoder signal from the master axis (which is not under PMAC's control) is fed into one of PMAC's encoder inputs. This master signal is typically either from an open-loop drive or a handwheel knob. Ix05 and Ix06 control this function.

5.10.5 - Cutter Radius Compensation

PMAC provides the capability for performing cutter (tool) radius compensation on the moves it performs. This compensation can be performed among the X, Y, and Z axes, which should be physically perpendicular to each other. The compensation offsets the described path of motion perpendicular to the path by a programmed amount. Cutter radius compensation is valid only in **LINEAR** and **CIRCLE** move modes. The moves must be specified by **F** (feedrate), not **TM** (move time). PMAC must be in move segmentation mode (I13 > 0) to do this compensation (I13 > 0 is required for **CIRCLE** mode anyway.). Program commands CC0, CC1, CC2, CCR and NORMAL control this feature.

5.10.6 - Synchronous M-Variable Assignment

The scan of a motion program and execution of the commands in it are governed by the lookahead feature. PMAC will calculate move commands ahead of time for a proper blending and will execute every instruction in between immediately. This ahead-of-time situation would make an M-variable assignment asynchronous to the motion profiles unless a double equal sign is used instead. M1==1, for example, will indicate PMAC that the assignment have to take place at the blending point between the previous move encountered and the next. In LINEAR and CIRCLE mode moves, this blending occurs V*TA/2 distance ahead of the specified intermediate point, where V is the commanded velocity of the axis, and TA is the acceleration (blending) time. This is only available for M-variables and that are not in the form TWB, TWD, TWR, TWS.

5.10.7 - Synchronizing PMAC to Other PMACs

When multiple PMACs are used together, inter-card synchronization is maintained by passing the servo clock signal from the first card to the others. With careful writing of programs, this permits complete coordination of axes on different cards.

5.10.8 - Axis Transformation Matrices

PMAC provides the capability to perform matrix transformation operations on the X, Y, and Z axes of a coordinate system. These operations have the same mathematical functionality as the matrix forms of the axis definition statements, but these can be changed on the fly in the middle of programs; the axis definition statements are meant to be fixed for a particular application. The matrix transformations permit translation, rotation, scaling, mirroring, and skewing of the X, Y, and Z axes. They can be very useful for English/metric conversion, floating origins, making duplicate mirror images, repeating operations with angle offsets, and more. The matrices gets implemented by the use of Q-variables and the commands DEFINE TBUF, TSEL, TINIT, ADIS, IDIS, AROT and IROT.

5.10.9 - Position-Capture and Position-Compare Functions

The position-capture function latches the current encoder position at the time of an external event into a special register. It is executed totally in hardware, without the need for software intervention (although it is set up, and later serviced, in software). This means that the only delays in the capture are the hardware gate delays (negligible in any mechanical system), so this provides an incredibly accurate capture function. The move-until-trigger functions (either jog or motion program) conveniently use the position capture feature for continuous motions until a trigger condition is reached.
The position-compare feature is essentially the opposite of the position-capture function. Instead of storing the position of the counter when an external signal changes, it changes an external signal when the counter reaches a certain position.

5.10.10 - Learning a Motion Program

It is possible to have PMAC “learn” lines of a motion program using the on-line **LEARN** command. In this operation, the axes are moved to the desired position and the command is given to PMAC. PMAC then adds a command line to the open motion program buffer that represents this position. This process can be repeated to learn a series of points.
The motors can be open-loop or closed-loop as they are moved around.

6.0 - PLC Programs

PMAC will stop the scanning of the motion program lines when enough move commands are calculated ahead of time. This feature is called "look-ahead" and it is necessary to properly blend the moves together and to observe the motion safety parameters. In the following example PMAC calculates up to the third move and will stop the program scanning until the first move is completed; that is, when more move planning is required:

Example:

```
OPEN PROG 1 CLEAR          ; Open program buffer
I13=0                      ; Two moves ahead of calculation
LINEAR INC TA100 TS0 F50 ; Mode commands
X1                         ; First Move
X1                         ; Second Move
X1                         ; Third Move
M1=1                       ; This line will be executed only after the first move is completed
CLOSE                      ; Close written buffer, program one
```

In contrast, enabled PLCs are continuously executed from beginning to end regardless of what any other PLC or Motion program is doing. PLCs are called *asynchronous* because are designed for actions that are asynchronous to the motion. Also, they are called PLC programs because they perform many of the same functions as hardware programmable logic controllers. PLC programs are numbered 0 through 31 for both the compiled and uncompiled PLCs. This means that you can have both a compiled PLC n and an uncompiled PLC n stored in PMAC. The faster execution of the compiled PLCs comes from two factors: first, from the elimination of interpretation time, and second, from the capability of the compiled PLC programs to execute integer arithmetic. However, the space dedicated to store up to 32 compiled PLC programs is limited to 15K (15,360) 24-bit words of PMAC memory; or 14K (14,336) words if there is a user-written servo as well. PLC programs 1-31 are executed in background. Each PLC program executes one scan (to the end or to an **ENDWHILE** statement) uninterrupted by any other background task (although it can be interrupted by higher priority tasks). In between each PLC program, PMAC will do its general housekeeping, and respond to a host command, if any. In between each scan of each individual background interpreted PLC program, PMAC will execute one scan of all active background compiled PLCs. This means that the background compiled PLCs execute at a higher scan rate than the background interpreted PLCs. For example, if there are seven active background interpreted PLCs, each background compiled PLC will execute seven scans for each scan of a background interpreted PLC. At power-on\reset PLCC programs run after the first PLC program runs. These are the suggested uses of all the available PLC buffers:

- **PLC0:** PLC program 0 is a special fast program that operates at the end of the servo interrupt cycle with a frequency specified by variable I8 (every I8+1 servo cycles). This program is meant for a few time-critical tasks, and it should be kept small, because its rapid repetition can steal time from other tasks. A PLC 0 that is too large can cause unpredictable behavior and can even trip PMAC's Watchdog Timer by "starving" background tasks of time to execute.
- **PLCC0:** the compiled PLCC0 should be used in the same cases as PLC0, taking advantage of the faster execution rate that a compiled PLC provides. Both PLC0 and PLCC0 could be defined at the same time.
- **PLC1:** This is the first code that PMAC will run on power-up, assuming that I5 was saved with a value of 2 or 3. This makes PLC1 the appropriate PLC to initialize parameters, perform commutated motors phase search and run motion programs. PLC1could also disable other PLCs before they start running and could disable itself at the end of its execution.
- **PLC2:** Since PLC1is suggested as an initialization PLC (and could potentially run only once on power-up), PLC2 is the first PLC in the remaining sequence from 2 to 31. This makes PLC2 the ideal place to copy digital input information from I\O expansion boards like the ACC-34 into its image variables. This way, PLCs 3 to 30 could use the input information, writing the necessary output changes to the outputs image variables.
- **PLC3 to PLC30:** PLC programs are particularly useful for monitoring analog and digital inputs, setting outputs, sending messages, monitoring motion parameters, issuing commands as if from a host, changing gains, and starting and stopping moves. By their complete access to PMAC variables and I/O and their asynchronous nature, they become very powerful adjuncts to the motion control programs.
- **PLCC3 to PLCC30:** compiled PLCs are convenient for its faster execution in compare to regular PLCs. Since the execution rate of compiled PLCs is the same as some of the safety checks (following error limits, hardware overtravel limits, software overtravel limits, and amplifier faults), PLCCs are ideal for replace or complement them. However, due to its limited allocated memory space, PLCCs should be reserved only for faster execution critical tasks.
- **PLC31:** this is the last executed PLC in the sequence from 1 to 31. PLC31 is then recommended for copying the output image variable (changed in lower number PLCs executed previously) into the actual outputs of an I\O expansion board like, for example, the ACC-34A.

6.1 - Entering a PLC Program

- PLCs are programmed in the same way as motion programs are, in a text editor window for later downloading to PMAC.
- Before start writing the PLC it is good practice to make sure that memory has not been tied up in data gathering or program trace buffers, by issuing **DELETE GATHER** and **DELETE TRACE** commands.
- Open the buffer for entry with the **OPEN PLC n** statement, where **n** is the buffer number. Next, if there is anything currently in the buffer that should not be kept, it should be emptied with the **CLEAR** statement (PLC buffers may not be edited on the PMAC itself; they must be cleared and re-entered). If the buffer is not cleared, new statements will be added onto the end of the buffer.
- When you are finished, you close the buffer with the **CLOSE** command. Opening a PLC program buffer automatically disables that program. After it is closed, it remains disabled, but it can be re-enabled again with the **ENABLE PLC n** command, where n is the buffer number (0--31). I5 must also be set properly for a PLC program to operate.
- At the closing, PMAC checks to make sure all IF branches and WHILE loops have been terminated properly. If not, it reports an error, and the buffer is inoperable. You should then correct the PLC program in the host and re-enter it (clearing the erroneous block in the process, of course). This process is repeated for all of the PLC buffers you wish to use.
- Because all PLC programs in PMAC's memory are enabled at power-on/reset it is good practice to have I5 saved as 0 in PMAC's memory when developing PLC programs. This will allow you to reset PMAC and have no PLC's running (an enabled PLC only runs if I5 is set properly) and more easily recover from a PLC programming error.

  Structure example:

```
CLOSE
DELETE GATHER
DELETE TRACE
OPEN PLC n CLEAR
      {PLC statements}
CLOSE
ENABLE PLC n
```

- To erase an uncompiled PLC program, you must open the buffer, clear the contents, then close the buffer again. This can be done with 3 commands on one line, as in:

```
OPEN PLC 5 CLEAR CLOSE
```

6.2 - PLC Program Structure

The important thing to remember in writing a PLC program is that each PLC program is effectively in an infinite loop; it will execute over and over again until told to stop. (These are called PLC because of the similarity in how they operate to hardware Programmable Logic Controllers -- the repeated scanning through a sequence of operations and potential operations.)

6.3 - Calculation Statements

Much of the action taken by a PLC is done through variable value assignment statements: **{variable}={expression}**. The variables can be I, P, Q, or M types, and the action thus taken can affect many things inside and outside the card. Perhaps the simplest PLC program consists of one line:

```
P1=P1+1
```

Every time the PLC executes, usually hundreds of times per second, P1 will increment by one.
Of course, these statements can get a lot more involved. The statement:

```
P2=M162/(I108*32*10000)*COS (M262/(I208*32*100))
```

could be converting radial (M162) and angular (M262) positions into horizontal position data, scaling at the same time. Because it updates this very frequently, whoever needs access to this information (e.g. host computer, operator, motion program) can be assured of having current data.

6.4 - Conditional Statements

Most action in a PLC program is conditional, dependent on the state of PMAC variables, such as inputs, outputs, positions, counters, etc. You may want your action to be level-triggered or edge-triggered; both can be done, but the techniques are different.

6.4.1 - Level-Triggered Conditions:

A branch controlled by a level- triggered condition is easier to implement. Taking our incrementing variable example and making the counting dependent on an input assigned to variable M11, we have:

```
IF (M11=1)
      P1=P1+1
ENDIF
```

As long as the input is true, P1 will increment several hundred times per second. When the input goes false, P1 will stop incrementing.

6.4.2 - Edge-Triggered Conditions:

Suppose instead that you only want to increment P1 once for each time M11 goes true (triggering on the rising edge of M11 sometimes called a "one-shot" or "latched"). To do this, we must get a little more sophisticated. We need a compound condition to trigger the action, then as part of the action, we set one of the conditions false, so the action will not occur on the next PLC scan. The easiest way to do this is through the use of a "shadow variable", which will follow the input variable value. Action is only taken when the shadow variable does not match the input variable. Our code could become:

```
IF (M11=1)
      IF (P11=0)
            P1=P1+1
            P11=1
      ENDIF
ELSE
      P11=0
ENDIF
```

Notice that we had to make sure that P11 could follow M11 both up and down. We set P11 to 0 in a level-triggered mode; we could have done this edge-triggered as well, but it does not matter as far as the final outcome of the routine is concerned, it is about even in calculation time, and it saves program lines.

6.5 - WHILE Loops

Normally a PLC program executes all the way from beginning to end within a single scan. The exception to this rule occurs if the program encounters a true **WHILE** condition. In this case, the program will execute down to the **ENDWHILE** statement and exit this PLC. After cycling through all of the other PLCs, it will re-enter this PLC at the **WHILE** condition statement, not at the beginning. This process will repeat as long as the condition is true. When the **WHILE** condition goes false, the PLC program will skip past the **ENDWHILE** statement and proceed to execute the rest of the PLC program.
If we want to increment our counter as long as the input is true, and prevent execution of the rest of the PLC program, we could program:

```
WHILE (M11=1)
      P1=P1+1
ENDWHILE
```

This structure makes it easier to "hold up" PLC operation in one section of the program, so other branches in the same program do not have to have extra conditions so they do not execute when this condition is true. Contrast this to using an IF condition (see above).

6.6 - COMMAND and SEND statements

One of the most common uses of PLCs is to start motion programs and Jog motors by means of command statements. Some **COMMAND** action statements should be followed by a **WHILE** condition to ensure they have taken effect before proceeding with the rest of the PLC program. This is always true if a second **COMMAND** action statement that requires the first **COMMAND** action statement to finish will follow. (Remember, **COMMAND** action statements are only processed during the communications section of the background cycle.) Suppose you want an input to stop any motion in a Coordinate System and start motion program 10. The following PLC could be used.

```
M187->Y:$0817,17,1              ; &1 In-position bit (AND of motors)
OPEN PLC3 CLEAR
IF (M11=1)                      ; input is ON
      IF (P11=0)                ; input was not ON last time
            P11=1               ; set latch
            COMMAND"&1A"        ; ABORT all motion
            WHILE (M187=0)      ; wait for motion to stop.
            ENDW
            COMMAND"&1B10R"     ; start program 10
      ENDIF
ELSE
      P11=0                     ; reset latch
ENDIF
CLOSE
```

Any **SEND**, **COMMAND**, or **DISPLAY** action statement should be done only on an edge-triggered condition, because the PLC can cycle faster than these operations can process their information, and the communications channels can get overwhelmed if these statements get executed on consecutive scans through the PLC.

```
IF (M11=1)                      ; input is ON
      IF (P11=0)                ; input was not ON last time
            COMMAND"#1J+"       ; JOG motor
            P11=1               ; set latch
      ENDIF
ELSE
      P11=0                     ; reset latch
ENDIF
```

6.7 - Timers

Timing commands like DWELL or DELAY are only reserved to motion programs and cannot be used for timing purposes on PLCs. Instead, PMAC has four 24-bit timers that you can write to, and count down once per servo cycle. These timers are at registers X:$0700, Y:$0700, X:$0701, and Y:$0701. Usually a signed M-variable is assigned to the timer; a value is written to it representing the desired time in servo cycles (multiply milliseconds by 8,388,608/I10); then the PLC waits until the M-variable is less than 0.

Example:

```
M90->X:$0700,0,24,S       ; Timer register 1 (8388608/I10 msec)
M91->Y:$0700,0,24,S       ; Timer register 2 (8388608/I10 msec)
M92->X:$0701,0,24,S       ; Timer register 3 (8388608/I10 msec)
M93->Y:$0701,0,24,S       ; Timer register 4 (8388608/I10 msec)
OPEN PLC3 CLEAR
M1=0                      ; Reset Output1 before start
M90=1000*8388608/I10      ; Set timer to 1000 msec, 1 second
WHILE (M90>0)             ; Loop until counts to zero
ENDWHILE
M1=1                      ; Set Output 1 after time elapsed
DIS PLC3                  ; disables PLC3 execution (needed in this example)
CLOSE
```

If you need more timers, probably the best technique to use is in memory address X:0. This 24-bit register counts up once per servo cycle. We will store a starting value for this, then each scan subtract the starting value from the current value and compare the difference to the amount of time we wish to wait.

Example:

```
M0->X:$0,24                  ; Servo counter register
M85->X:$07F0,24              ; Free 24-bit register
M86->X:$07F1,24              ; Free 24-bit register
OPEN PLC 3 CLEAR
M1=0                         ; Reset Output1 before start
M85=M0                       ; Initialize timer
M86=0
WHILE(M86<1000)              ; Time elapsed less than specified time?
    M86=M0-M85
    M86=M86*I10/8388608      ; Time elapsed so far in milliseconds
ENDWHILE
M1=1                         ; Set Output 1 after time elapsed
DISABLEPLC3                  ; disables PLC3 execution (needed in this example)
CLOSE
```

Even if the servo cycle counter rollovers (start from zero again after the counter is saturated), by subtracting into another 24-bit register, we handle rollover gracefully.

Rollover example:

M0 = 1000
M85 = 16777000
M86 = 1216

Bit	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0
M85	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	0	0	0
M86	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0

← Carry-out bit

6.8 - Compiled PLC Programs

PLCCs are compiled by PEWIN in the downloading process. Only the compiled code gets downloaded to PMAC. Therefore it is suggested to save the ASCII source code in the host computer separately since it cannot be retrieved from PMAC. Compiled PLCs are firmware dependent and so they must be recompiled when the firmware is changed in PMAC. If more than one PLCC is programmed, all the PLCCs code must belong to the same ASCII text file. PEWIN will compile all the PLCC code present on the file and place it in the appropriate buffer in PMAC. If a single PLCC code is downloaded, all the rest PLCCs that might have been present in memory will be erased, remaining only the last compiled code.
The multiple-file download feature of the PEWIN File menu allows having the PLCC codes in different files. They will be combined by PEWIN in the downloading process.
The use of L-variables in a PLC program statement is the sign to the compiler that the statement is to be executed using integer operations instead of floating-point operations.
To implement integer arithmetic in a compiled PLC, the user must both define any L-variables to be used and substitute them in the programs for the variables that were used in the interpreted form (usually M-variables). The compiler will interpret statements containing only L-variables (properly defined) and integer constants as operations to be executed using integer arithmetic in compiled PLCs. Preparation of compiled PLCs is a multi-step process, the basic steps are as follows:

1) Write and debug the PLC programs in interpreted form (simple PLCs programs).
2) Change all references to PLCs to be compiled from **PLC** to **PLCC**.
3) For integer arithmetic, define L-variables and substitute these for the old variable names in the programs.
4) Combine all of the PLC programs to be compiled into one file on the PC.
5) Make sure the “Support MACRO’s/PLCCs” option is checked before downloading.
6) Activate the compiled PLCs. If operation is not correct, return to step 1 or 2.
7) PLCCs could be deleted using the DELETE PLCCn command (replace n by the appropriate number).

7.0 - Troubleshooting Section

PMAC is a highly reliable device and has several safety mechanisms to prevent continuous damage and malfunctions. When PMAC shuts-down or an erratic behavior is observed the following reset procedure should be tried.

7.1 - Resetting PMAC to factory defaults

1) If PMAC is communicating with the host computer skip steps 2-7 on this list.
2) Turn off PMAC or the host computer where PMAC is installed.
3) Remove all cables connected to PMAC only leaving connected the serial port and power cables if present.
4) Using the appropriate hardware reference for the particular PMAC in question check that all its jumpers are at the default configuration or properly changed to accommodate the particular setup for the machine. Make sure that jumper E50 is properly installed; otherwise any SAVE command issued to PMAC will not have any effect.
5) Place the jumper E51 in PMAC (1) or jumper E3 on PMAC2. This is a hardware re-initialization jumper.
6) After power-up try establishing communications again with a reliable software package like the PEWIN program provided by Delta Tau.
7) On power-up, with the re-initialization jumper installed, some PMAC's with the flash memory option will be in a mode called "bootstrap". This means that will accept a binary file downloaded to change its internal firmware. If this is the case, follow the instructions on the PEWIN screen to disable the downloading process (usually pressing CTRL+R).
8) Try communications with PEWIN and type the following commands when the terminal gets successfully open (follow the communications troubleshooting section below in case communications are still not established):

```
$$$***                      ;Global Reset
P0..1023=0                  ;Reset P-variables values
Q0..1023=0                  ;Reset Q-variables values
M0..1023->* M0..1023=0      ;Reset M-variables definitions and values
UNDEFINE ALL                ;Undefine Coordinate Systems
SAVE                        ;Save this initial, "clean"configuration
```

9) If the re-initialization jumper was installed remove it at this time. Restore PMAC in the computer and power it up.
10) Try communications again and configure PMAC for your application. It is strongly recommended to have a backup file saved in the host computer with all the parameters and programs that PMAC needs to run the application. Furthermore, since the host computer could also fail and be replaced, save the configuration file both in the host computer and in a floppy disk stored in a safe place. This file must be downloaded and a SAVE command must be issued to PMAC.

7.2 - The watchdog timer (red LED)

The PMAC motion control board has an on-board "watchdog timer" (sometimes called a "dead-man timer" or a "get-lost timer") circuit whose job it is to detect a number of conditions that could result in dangerous misfunction, and shut down the card to prevent a misfunction. The philosophy behind the use of this circuit is that it is safer to have the system not operate at all than to have it operate improperly.
Because the watchdog timer "wants" to fail, and many components of the board, both hardware and software, must be working properly to keep it from failing, it may not be immediately obvious what the cause of a watchdog timer failure is.
The hardware circuit for the watchdog timer requires that two basic conditions be met to keep it from tripping. First, it must see a DC voltage greater than approximately 4.75V. If the supply voltage is below this value, the circuit's relay will trip. This prevents corruption of registers due to insufficient voltage. The second necessary condition is that the timer must see a square wave input (provided by the PMAC software) of a frequency greater than approximately 25 Hz. If the card, for whatever reason, due either to hardware or software problems, cannot set and clear this bit repeatedly at this frequency or higher, the circuit's relay will trip.
Every RTI, PMAC reads the 12-bit watchdog timer register (Y register $1F) and decrements the value by 8 -- this toggles bit 3. If the resulting value is not less than zero, it copies the result into a register that forces the bit 3 value onto the watchdog timer. Repeated, this process provides a square-wave input to the watchdog timer.

In the background, PMAC executes one scan through an individual PLC program, then checks to see if there are any complete commands, responding if there are, then executes the housekeeping functions. This cycle is repeatedly endlessly. Most of the housekeeping functions are safety checks such as following error limits and overtravel limits. When it is done with these checks, PMAC sets the 12-bit watchdog timer register back to its maximum value. As long as this occurs regularly at least every 512 RTI cycles, the watchdog timer will not trip.
The purpose of this two-part control of the timer is to make sure all aspects of the PMAC software are being executed, both in foreground (interrupt-driven) and background. If anything keeps either type of routine from executing, the watchdog will fail quickly.
The only recover for this failure, assuming the 5 Volts power supply is satisfactory, is to hardware reset PMAC.

7.3 - Establishing communications

Either the Executive or Setup program can be used to establish initial communications with the card. Both programs have menus that allow you to tell the PC where to expect to find the PMAC and how to communicate with it at that location. If you tell it to look for PMAC on the bus, you must also tell it PMAC's base address on the bus (this was set up with jumpers on PMAC). If you tell it to look for PMAC on a COM port, you must tell it the baud rate (this was set up with jumpers or switches on the PMAC).
Once you have told the program where and how to communicate with PMAC, it will attempt to find PMAC at that address by sending a query command and waiting for the response. If it gets the expected type of response, it will report that it has found PMAC, and you will be able to proceed.
If it does not get the expected type of response after several attempts, it will report that it has not found PMAC, check the following:

7.3.1 - General

1. Is the green LED (power indicator) on PMAC's CPU board ON, as it should be? If it is not, find out why PMAC is not getting a +5V voltage supply.
2. Is the red LED (watchdog timer indicator) on PMAC's CPU board OFF, as it should be? If it is ON, make sure PMAC is getting very close to 5V supply -- at less than 4.75V, the watchdog timer will trip, shutting down the card. The voltage can be probed at pins 1 and 3 of the J8 connector (A1 and A2 on the PMAC-VME). If the voltage is satisfactory, inspect PMAC to see that all inter-board connections and all socketed ICs are well seated. If you cannot get the card to run with the red LED off, contact the factory.

7.3.2 - Bus Communications

1. Do the bus address jumpers (E91-E92, E66-E71) set an address that matches the bus address that the Executive program is trying to communicate with?
2. Is there something else on the bus at the same address? Try changing the bus address to see if communications can be established at a new address. Address 768 (300 hex) is usually open.

7.3.3 - Serial communications

1. Are you using the proper port on the PC? Make sure if the Executive program is addressing the COM1 port, that you have cabled out of the COM1 connector.
2. Does the baud rate specified in the Executive program match the baud rate setting of the E44-E47 jumpers on PMAC?
3. With a breakout box or oscilloscope, make sure you see action on the transmit lines from the PC as you type into the Executive program. If you do not, there is a problem on the PC end.
4. Probe the return communication line while you give PMAC a command that requires a response (e.g. **<CONTROL-F>**). If there is no action, you may have to change jumpers E9-E16 on PMAC to exchange the send and receive lines. If there is action, but the host program does not receive characters, you may have RS-232 receiving circuitry that does not respond at all to PMAC's RS-422 levels. If you have another model of PC, try using it as a test (most models accept RS-422 levels quite well). If you cannot get your computer to accept the signals, you may need a level-conversion device, such as Delta Tau's Accessory-26.

7.4 - Motor parameters

1. No movement at all. Check the following:
   a. Are both limits held low to AGND and sourcing current out of the pins?
   b. Do you have proper supply to A+15V, A-15V, and AGND?
   c. Is your proportional gain (Ix30) greater than zero?
   d. Can you measure any output at the DAC pin when an O command has been given?
   e. Are you tripping your following error limit? Increase the fatal following error limit (Ix11) by setting it to a more appropriate value, and try to move again.
2. Movement, but sluggish. Check the following:
   a. Is proportional gain (Ix30) too low? Try increasing it (as long as stability is kept).
   b. Is your "big step limit" (Ix67) too low? Try increasing it to 8,000,000 -- near the maximum -- to eliminate any effect.
   c. Is your output limit (Ix69) too low? Try increasing it to 32,767 (the maximum) to make sure PMAC can output adequate voltage.
   d. Can an integrator help? Try increasing integral gain (Ix33) to 10,000 or more, and the integration limit (Ix63) to 8,000,000.
3. Runaway condition. Check the following:
   a. Do you have feedback? Check that you can read position changes in both directions.
   b. Does your feedback polarity match output polarity? Recheck the polarity match as explained above.
4. Brief movement, then stop. Check the following:
   a. Are you tripping your following error limit? Increase the fatal following error limit (Ix11) by setting it to a more appropriate value, and try to move again.

If you are holding position well, but cannot move the motor, you probably do not have your hardware limits held low. Check which limits I125 is addressed to (usually +/-LIM1), then make sure those points are held low (to AGND), and sourcing current (unscrew the wire from the terminal block and put your ammeter in series with this circuit if you need to confirm this). Refer to the section *"Installing and Configuring PMAC"* for details on checking the limit inputs.
If your motor "dies" after you give it a jog command, you have probably exceeded your fatal following error limit. If this has happened, it is either because you have asked for a move that is more than the system can physically do (if so, reduce I122), or because you are very badly tuned (if this is the case, you will need to increase proportional gain I130). To restore closed-loop control, issue the **J/** command.

7.5 - Motion programs

If the program does not run at all, there are several possibilities:

1. Can you list the program? In terminal mode, type **LIST PROG 1** (or whichever program), and see if it is there. If not try to download it to the card again.
2. Did you remember to close the program buffer? Type **A** just in case the program is running; type **CLOSE** to close any open buffer; type **B1** (or your program #) to point to the top of the program; and type **R** to try to run it again.
3. Can each motor in the coordinate system be jogged in both directions? If not, review that motor's setup.
4. Have any motors been assigned to the coordinate system that is not really set up yet? Every motor in the coordinate system must have its limits held low, even if there is no real motor attached.

Try the following steps for any other motion program problem:

1) Type "`&1%100`" in the terminal window.
2) Check that you can appropriately Jog only one of the motors that you intend to use in your motion program.
3) Type the following commands in a text editor to be downloaded to PMAC:

```
close                   ; Close any buffer opened
delete gather           ; Erase unwanted gathered data
undefine all            ; Erase coordinate definitions in all coordinate systems

#1->2000X               ; Replace #1 for the motor you want to use and 2000 by the
                        ; appropriate scale factor for the number of counts per user units
```

```
OPEN PROG 1 CLEAR       ; Prepare buffer to be written
LINEAR                  ; Linear interpolation
INC                     ; Incremental mode
TA500                   ; Acceleration time is 500 msec
TS0                     ; No S-curve acceleration component
TM2000                  ; Total move time is 500 + 2000=2500 msec
X1                      ; One unit of distance, 2000 encoder counts
CLOSE                   ; Close written buffer, program one
```

4) To run it, press CTRL+A and then type `B1R` in the terminal window.
5) Repeat steps 2 through 4 for all the motors that you intend to run in your actual motion program.

A good method to test motion programs is to run them at lower than one hundred percent override rate. Any value for n from 1 to 100 in the "%n" online command will run the motion programs slower, increasing the chances of success of execution. For example, in the terminal window type: `&1 %75 B1R`
If a program run successfully at lower feedrate override values there could be mainly two reasons why it fails at 100%: either there is insufficient calculation time for the programmed moves or the acceleration and\or velocity parameters involved are unsuitable for the machine into consideration. Look for further details in the section entitled "*PMAC Tasks*".

7.6 - PLC programs

PLCs and PLCCs are one of the most common sources for communication or watchdog timer failures.

- Any **`SEND`**, **`COMMAND`**, or **`DISPLAY`** action statement should be done only on an edge-triggered condition, because the PLC can cycle faster than these operations can process their information, and the communications channels can get overwhelmed if these statements get executed on consecutive scans through the PLC.

```
IF (M11=1)                      ; input is ON
      IF (P11=0)                ; input was not ON last time
            COMMAND"#1J+"       ; JOG motor
            P11=1               ; set latch
      ENDIF
ELSE
      P11=0                     ; reset latch
ENDIF
```

- PLC0 or PLCC0 are meant to be used for only a very few tasks (usually a single task) that must be done at a higher frequency than the other PLC tasks. The PLC 0 will execute every real-time interrupt as long as the tasks from the previous RTI have been completed. *PLC 0 is potentially the most dangerous task on PMAC as far as disturbing the scheduling of tasks is concerned. If it is too long, it will "starve" the background tasks for time.* The first thing you will notice is that communications and background PLC tasks will become sluggish. In the worst case, the watchdog timer will trip, shutting down the card, because the housekeeping task in background did not have the time to keep it updated.

- Because all PLC programs in PMAC's memory are enabled at power-on/reset it is good practice to have I5 saved as 0 in PMAC's memory when developing PLC programs. This will allow you to reset PMAC and have no PLC's running (an enabled PLC only runs if I5 is set properly) and more easily recover from a PLC programming error.

- As an example, type these commands in the terminal window. After that, open a "watch" window and monitor for P1 to be counting up:

```
OPEN PLC1 CLEAR         ; Prepare buffer to be written
P1=P1+1                 ; P1 continuously incrementing
CLOSE                   ; Close written buffer, PLC1
I5=2
```

Press <`CTRL+D`> and type `ENA PLC1`

USA
Delta Tau Data Systems Inc
21314 Lassen St.
Chatsworth, CA 91311
U.S.A.
PH: (818) 998-2095
FAX: (818) 998-7807
E-MAIL: support@deltatau.com

Europe
Delta Tau Europa
Rheinweg 4 CH-8200
Schaffhausen, Switzerland
Tel: +41-52-625-2088
Fax: +41-52-625-4482
E-mail: bradp@deltatau.com

PMAC-Japan
13-10, Nihonbashi- Odennmacho,
Chuo-ku,
Tokyo, 103 Japan
PH: 03-3665-6421
FAX: 03-3665-6888
E-MAIL: info@pmac-j.com

South Korea
Delta Tau Intl Korea
Hyundai Apt. 1103-1205
1575-4 Ilsan2, Ilsan, Koyang,
Kyungki-do, South Korea, 411-312
PH: +82-344-975-6156
FAX: +82-344-957-6155
E-MAIL: jypark@bora.dacom.co.kr